Litzkendorf / Onnen

GFA BASIC

Tips & Tricks

EIN DATA BECKER BUCH

Danksagung

Anfang 2019 begann ich damit, alles an Atari-Hardware zu erwerben, was ich bereits in den 80er/90er-Jahren schon einmal besaß. So auch Bücher, die sich speziell mit dem Thema Programmierung beschäftigen. Ich stellte schnell fest, dass dieses Vorhaben heutzutage schwieriger werden könnte als der Kauf eines Atari ST. Für eine gebrauchte Ausgabe von „Das große GFA Basic 3.5 Buch“ zahlte ich mehr als den damaligen Neupreis! Mein Wunsch war aber, genau dieses Buch wieder in meiner Sammlung zu sehen. Da das Buch sehr gut erhalten war, mochte ich es auch so belassen und scannte es daher ein. Somit konnte ich nun bequem digital darin blättern, nach Begriffen suchen und Lesezeichen erstellen. Später entstand allerdings der Gedanke, dass diese digitale Version doch eigentlich der Allgemeinheit zur Verfügung gestellt werden sollte. Eine Anfrage zur Genehmigung beim Data Becker Verlag war allerdings nicht mehr möglich, da der Verlag seinen Geschäftsbetrieb zum 31.03.2014 aufgegeben hatte und es auch keine Rechtsnachfolger gibt. Somit machte ich mich auf die Suche nach dem Autor, Uwe Litzkendorf, und fand ihn auch. Mit großer Freude durfte ich feststellen, dass er meine Meinung teilte. Und mehr noch, Uwe machte den Vorschlag, dass ich es nicht nur bei diesem einem Buch belassen sollte, sondern auch weitere seiner geschriebenen Fachbücher zum Thema GFA-Basic digitalisieren dürfte. Ich war begeistert und erklärte mich gerne bereit dieses Projekt durchzuführen. So entstand aus einer einfachen E-Mail Anfrage eine wunderbare Möglichkeit den Inhalt dieser Bücher digital für die Zukunft zu erhalten und allen Interessierten kostenfrei zur Verfügung stellen zu können. Dafür möchte mich bei Uwe noch einmal sehr herzlich bedanken.

Thomas Werner
Bearbeitungsstand: 03. Mai 2020

Neues Vorwort zur Digitalisierung

Es hat lange gedauert! Aber nun hat sich Thomas Werner die viele Arbeit gemacht, den größten Teil meiner Bücher nach, zum Teil über dreißig, Jahren in interaktiver PDF-Form zu digitalisieren und als Public Domain kostenlos online zur Verfügung zu stellen. Nachdem meine Bücher einen langen Dornröschenschlaf hinter sich haben, sollen sie und das immense darin festgeschriebene Wissen jetzt wieder zum nützlichen Leben erweckt werden :O)

Meiner Ansicht nach ist die objektorientierte Programmierung nicht für die Massenanwendung geeignet. Nach meiner Berechnung sind nur ca. 15 Prozent der Bevölkerung in der Lage, mit der OOP sinnvolle Erfolge zu erzielen. Um aber "massenfähig" zu sein, muss eine Programmiersprache mindestens 80 Prozent der Bevölkerung erreichen, damit sich Lehrer und Schüler auch außerhalb von IT-Leistungskursen über die Softwareentwicklung so verständigen können, dass der eine – vermittelbar und auch prüfbar – wenigstens "einigermaßen" versteht, was der andere meint. Andererseits "reißt unweigerlich der Faden" zwischen Ausbildern und Lernenden. Daher ist es auch kein Wunder, dass sich die prozeduralen und damit auch leicht anwendbaren Basic-Programmiersprachen, wie z.B. das sehr populäre GFA-Basic, einer gewissen Renaissance erfreuen.

Diese Tendenz werde ich natürlich als ehemals populärer Bestseller-Autor nach all meinen Möglichkeiten tatkräftig unterstützen. Ich verfüge über mehr als 4000 Seiten umfassenden, ausführlichen und auch unter den wachsamen Augen der Öffentlichkeit profund geprüften Software-Wissens.

Dieses Wissen ist auch in der heutigen Zeit absolut nicht überflüssig und veraltet, sondern bildet auch heute noch die Basis für algorithmisches Grundlagenwissen.

Aber damit nicht genug: ich habe zudem beschlossen, eine neue Programmiersprache namens "QS!X©" zu entwickeln. Sie wird sich in vielen Punkten an einfachem Standard-Basic anlehnen. Auf der weltweit überall auf allen Betriebssystemen und in jedem Standardbrowser verfügbaren – und damit 100% cross-kompatiblen – Plattform von HTML5/Canvas wird "QS!X©" als Open Source-Version (ähnlich LINUX) verfügbar sein. Nähere Informationen dazu finden Sie unter:

http://www.litzkendorf.net/invitation_info_d.pdf

Damit ist auch der "Klasse für die Masse"-Philosophie von Frank Ostrowski (dem GFA-Basic-Vater) Rechnung getragen. Wenn denn alles so funktioniert, wie ich es mir vorstelle, wird die weise, sanfte und erzfreundlich geduldige und bescheidene Denkart von Frank Ostrowski auch Jahre nach seinem (viel zu frühen) Ableben noch weltweit merkliche Wirkung tragen! Er bildet dann verdientermaßen – zumindest im IT-Business – die philosophische Grundlage für eine Art "Weltsprache"! Das würde ihm sicher sehr gefallen!

In treuem Gedenken an einen wirklich großen Mann, mit dem ich das Vergnügen hatte, teil- und zeitweise recht eng und vertraut zusammen zu arbeiten und dem mit "QS!X©" auch ein digitales – und verdientes – Denkmal gesetzt wird!

Uwe Litzkendorf
Hildesheim, im Mai 2020

Bei der Nutzung und Weitergabe der vorliegenden digitalen Version ist Folgendes zu beachten:

Ein Weiterverkauf der digitalen Ausgabe ist nicht gestattet. Die Rechte liegen weiterhin beim Autor.

Eine Weitergabe dieses PDFs ist nur in unveränderter Form erlaubt

Ausdrucke einzelner Seiten sind für rein private Zwecke selbstverständlich gestattet.

Öffentliche Vorführungen – auch Auszugsweise – sind gestattet, solange diese keinen finanziellen Zwecken dienen. Ausgenommen davon sind Presseberichterstattungen.

Auch wenn dieses Buch kostenfrei zur Verfügung gestellt wurde, hat die Erstellung einiges an privater Zeit, Geld und Arbeit gekostet. Wer dies zu schätzen weiß, darf sich gerne erkenntlich zeigen.

Weitere Informationen und eBooks finden sich unter:
http://ebook.pixas.de

ISBN 3-89011-193-9

Copyright © 1987 DATA BECKER GmbH
Merowingerstraße 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Vorwort

Nach dem Flop mit dem ursprünglichen Atari-BASIC und einer langen Zeit des Wartens auf eine Alternative kam die frohe Kunde: Es gibt ein neues BASIC - GFA-BASIC, Autor Frank Ostrowski. Sehr schnell stellte sich bei uns Programmierfreaks heraus, daß hier weit mehr als das Bekannte und Herkömmliche auf dem Bildschirm zu erleben war. Die Geschwindigkeit des Interpreters ist so groß, daß BASIC sogar für Profis interessant wurde. In Kombination mit dem Compiler wird man wohl kaum noch Programmieraufgaben finden, die nach noch mehr Schnelligkeit verlangen. Die Chance des strukturierten Programmierens schafft in unseren Quellcodes eine Übersichtlichkeit und Nachvollziehbarkeit, die bisher nur von Pascal, 'C' usw. bekannt war, Spaghetticode, jahrelang das Markenzeichen von BASIC, ist out. Und die Menge der Befehle und Routinen, die Frank Ostrowski implementiert hat, verlangt schon nach einer gehörigen Einarbeitungszeit, um all die vielen Möglichkeiten ausschöpfen zu können. Kurz: GFA-BASIC hat einen neuen Standard bei den Programmiersprachen gesetzt.

In diesem Buch geht es um das Ausschöpfen der Möglichkeiten. GFA-BASIC kann viel mehr, als auf den ersten Blick ersichtlich ist, viele Routinen, die in älteren BASIC-Varianten gar nicht, auf Umwegen oder sehr lang programmiert werden mußten, können hier 'mit links' geschaffen werden. Und mit dem nötigen Wissen über den Atari, sein Betriebssystem und seine versteckten Adressen werden virtuose Programme, geschrieben in BASIC, nicht nur von ein paar Profis, sondern von all denen, die es wissen wollen, kreiert werden können.

Schauen Sie zwei langgedienten Programmierern über die Schulter, die sich in GFA-BASIC verliebt haben.

Uwe Litzendorf hat schon zwei Monate nach Markteinführung von GFA-BASIC sein 'Großes GFA-BASIC-Buch' herausgebracht, das zum einen ein Handbuchersatz für viele wurde und zum anderen eins der ersten Malprogramme vorstellte, das viel mehr Grafikmöglichkeiten des Atari nutzte, als bisher be-

kannt waren. Außerdem ist er einer der beiden Autoren des DATA BECKER Führer 'GFA-BASIC'. Er hat sich vor allem auf die Betriebssystemroutinen und die dokumentierten und undokumentierten Adressen gestürzt und zeigt Möglichkeiten auf, die bisher z.T. noch nicht einmal bekannt waren.

Udo Onnen, seit Jahren professioneller EDV-Anwender und Programmierer, ist seit längerem als Autor von Zeitschriftenartikeln bekannt, die Problemlösungen in GFA_BASIC vorstellen und erläutern. Sein Interesse liegt vor allem in den Möglichkeiten der grafischen Benutzeroberfläche GEM und den Chancen, die sich durch die Benutzung von Fenstern, Objekten und Bildern für die problemlose Handhabung der Programme bieten.

Es bleibt zu hoffen, daß die Tips und Tricks, die hier aufgelistet sind, ihren Zweck erfüllen, nämlich auch Ihnen zu ermöglichen, ohne unnötige Umwege das Ziel, das Sie sich gesetzt haben, zu erreichen. Verzeihen Sie Fehler, die trotz sorgfältiger Korrektur immer noch auftreten können, aber behalten Sie diese nicht für sich.

Uwe Litzkendorf
Udo Onnen

Hildesheim/Hannover,
im Juni 1987

Inhaltsverzeichnis

1. Dem Computer über die Schulter geschaut 13

1.1. Der MEMORY-Spion 14
1.2 Tischlein deck' dich 31
1.3 Images zum Selbermachen - ein Image-Editor 41
1.4 Öfter mal was Neues 48
1.4.1 Alert-Icons austauschen 48
1.4.2 Maus-Icons austauschen 52
1.4.3 Aus Data mach' PUT 56
1.5 Spezial-Adressen 65
1.6 Atari-gesicherte System-Adressen 74

2. Das Floppy-ABC 85

3. TOS-Allerlei 113

3.1 XBIOS-Funktionen 115
3.2 GEMDOS-Funktionen 140
3.3 BIOS-Funktionen 150
3.4 VDISYS-Routinen 154
3.5 GEMSYS-Routinen 165

4. Objekte, Formulare und Resources 171

4.1 Objektgenerierung in GFA-BASIC 171
4.1.1 Die OBJECT-Struktur 174
4.1.2 Ob_spec-Varianten 179
4.1.3 Die TEDINFO-Struktur 181
4.2 Objekte zeichnen sich selbst 185
4.3 Resource Construction Set-Spielereien 186
4.3.1 Header-Files aus RCS nach BASIC konvertieren 189
4.4 Resources-Files laden und freigeben 191
4.5 Statusfragen 194
4.5.1 Ob_state 195

4.6 Formulare 198
4.7 Imageaufbesserungen 202
4.7.1 Die BITBLK-Struktur 203
4.7.2 Der Image-Generator für die Bilddaten 207
4.8 Neue Icons für unsere Programme 209
4.8.1 Die ICONBLK-Struktur 210
4.8.2 Ein Icon-Editor für die Objektstruktur ICON 214
4.9 Die Krönung: Das private Desktop 222
4.10 Die Bibliothek des Grafen 225
4.11 Menü mit mehreren Gängen 229
4.12 Fensterln 233
4.12.1 Sesam öffne dich 236
4.12.2 Fensterbewegungen 237
4.12.3 REDRAW 244
4.12.4 Slider und Arrows 249
4.12.5 Sieben Fenster? Das geht doch gar nicht! 255
4.13 Tutti-Frutti: Algorithmen für einen Resources-Baukasten 258
4.13.1 Objekte bewegen 260
4.13.2 Objekte definieren 261
4.13.3 Objekte spezifizieren 264
4.13.4 Der Resources-Kopf 268
4.13.5 Objekte speichern 270

5. Spezialitäten aus Kraut und Rüben 275

5.1 FORM_INPUT, etwas anders 275
5.2 Automatischer Zeilenumbruch für Textverarbeitung 280
5.3 Druckeranpassung statt Deskaccessory 281
5.4 Uhrzeit aus dem AUTO-Ordner 283
5.4.1 Die VT52-Bildschirmsteuerung 285
5.5 Und noch 'ne Uhr 287
5.6 Ein bißchen Perspektive gefällig 288
5.6.1 Der Kubus 290
5.6.2 Das Tortenstück 292
5.6.3 Zeichenroutine für perspektivische Diagramme 294
5.7. Variablen-Referenz 299
5.8 Der verboxte Screen 302

5.9 "Kalkuliere....." 310
5.10 Extension- und Backup-Service 321
5.11 Hochleistungskompressor 323
5.12 Alles Schiebung 328
5.13 Gut sortiert 332
5.14 GFA-BASIC-Erweiterung:
Sortier-Funktion per Monitor() 334

Anhang A: TOS- und GEM-Routinen-Verzeichnis 343

Anhang B: Stichwortverzeichnis 347

Hinweis

Bei den vorgestellten Programmen erklären wir die einzelnen Prozeduren nacheinander. Die Programmabschnitte sind nicht einzeln, sondern nur im Gesamten lauffähig. Die Programme befinden sich unter den im Programmkopf angegebenen Namen auf der beiliegenden Programmdiskette.

1. Dem Computer über die Schulter geschaut

Von jeher hat uns brennend interessiert, wie es in einem Computer-Speicher aussieht. Daß der Speicher eine riesige Ansammlung von Bits und Bytes ist, wird den meisten der Leser wohl bekannt sein. Aber was passiert darin eigentlich?

Um sich dieses geniale Durcheinander einmal genauer anzuschauen und den Computer (besser: das Betriebssystem) bei der Arbeit zu beobachten, gibt es im GFA-BASIC zwei Möglichkeiten.

Die erste ist die, daß man mit dem 'BMOVE'-Befehl einzelne Speicherbereiche ganz schnell hintereinander in den Bildschirmspeicher kopiert. Die zweite ist etwas einfacher, dafür aber überhaupt nicht variabel. Nämlich einfach den Bildschirmspeicher über den Speicherbereich, den man sehen möchte, 'drüberstülpen', also den Speicherbereich, den man auf dem Bildschirm sehen kann, in den gewünschten Bereich verlegen.

Das ist deshalb recht uneffektiv, weil man nun tunlichst vermeiden sollte, Bildschirmausgaben zu produzieren, da dann z.B. eine Box direkt in den Speicher 'gezeichnet' und die dort liegenden Bits zerstören würde. Das eröffnet natürlich auch die Möglichkeit, ganze Speicherbereiche zu invertieren, zu 'Xor'en oder zu 'Or'en, indem man z.B. im Graphmode 3 eine schwarze Pbox über den Bereich zeichnet (XOR) oder ein Speicherrechteck mit 'GET X,Y,X1,Y1,A$' zwischenspeichert und mit 'PUT X2,Y2,A$,7' an die gewünschte Speicherstelle transportiert (OR). Bei richtiger Handhabung ist dieser Vorgang sogar in manchen Beziehungen effektiver als der 'BITBLT'-Befehl. Wie dieses Verschieben des Bildschirmspeichers funktioniert, finden Sie unter 'XBIOS(5)' beschrieben.

1.1. Der MEMORY-Spion

Das folgende kleine (?!) Programm verwendet mehrere Tips und Tricks, die entweder im Listing oder an anderer Stelle beschrieben werden. Wenn Sie das Programm starten, können Sie mit Druck auf die <SPACE>-Taste den zweiten Effekt (Bildchirmspeicher verschieben) auslösen, wobei allerdings sonst keine weiteren Funktionen aktiv sind. Nach Druck auf eine beliebige Taste kehrt das Programm wieder in den Normal-Modus zurück (BMOVE in den Bildschirmspeicher). Während im Normal-Modus der Bmove-Befehl und die sonstigen Bildschirmanzeigen etwas Zeit brauchen, ist im anderen Modus (SPACE-Taste) der Speicher in Real-Time zu sehen, da absolut keine Zeitverzögerung zwischen den System-Aktionen und der Sichtbarkeit auf dem Bildschirm auftritt.

Eine weitere Hilfs-Funktion kann im Normal-Modus durch Druck auf die <Help>-Taste aktiviert werden. Während sonst in der unteren Bit-, Wert- und ASCII-Anzeige immer der Inhalt des Bytes angezeigt wird, auf welchem sich der Bytezeiger gerade befindet, wird nun nur noch die Adresse angezeigt, die beim Drücken der <Help>-Taste aktuell war. Diese Funktion wird wieder ausgeschaltet, indem noch einmal die <Help>-Taste gedrückt wird.

Im übrigen sind die Programmfunktionen jeweils in der Kopfzeile angezeigt. Im Edit-Modus haben Sie die zusätzliche Möglichkeit, mit Druck auf die rechte Maustaste den 'SNAP'-Modus aufzurufen, womit Sie beliebige Bildschirmausschnitte im GET/PUT-Format auf Diskette speichern können. Sie verlassen dieses Programm mit der <Esc>-Taste.

Viel Spaß mit diesem Programm, das Ihnen hoffentlich den richtigen 'Durchblick verschafft.

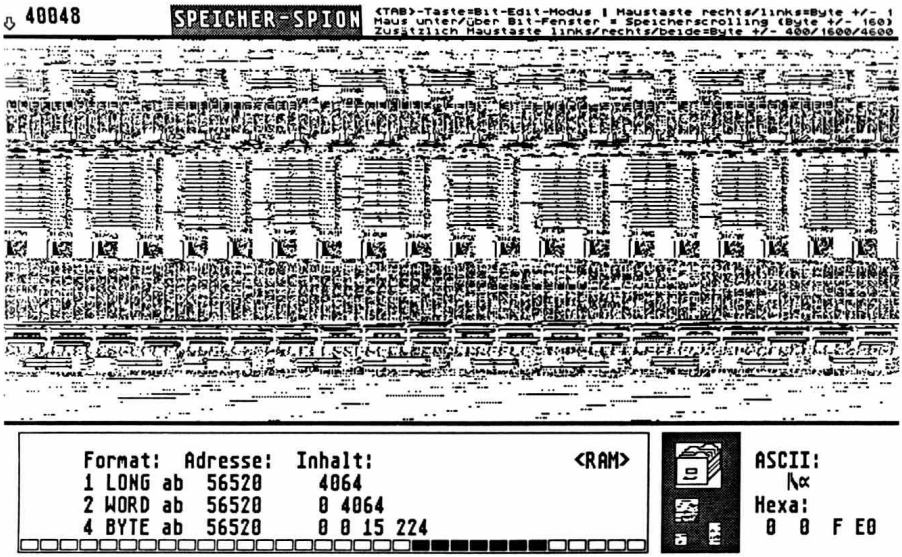


Abb. 1: Durchblick

```
' Programm: MEMO_SPY.BAS
'
' |''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''|
' |                    MEMORY - SPION                           |
' |--------------------------------------------------------------|
' ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

On Break Cont                        ! Abbruch-Funktion ausschalten
Ssp%=@Super(0)                       ! User-Modus ausschalten, ab jetzt
'                                    ! sind wir im Supervisor-Modus
Dim Hex_buff$(4)                     ! 4 Puffer für 4 Hexa-Bytestrings
Ram_top%=Lpeek(1070)                 ! letzte RAM-Adresse feststellen
'                                    !     (s. unter 'System-Adressen')
Rom_bot%=Lpeek(1266)                 ! erste ROM-Adresse feststellen
@Init                                ! Screen aufbauen
Hidem                                ! Maus aus
@Main                                ! zum Hauptprogramm
Showm                                ! Maus an
Usp%=@Super(Ssp%)                    ! Supervisor-Stack restaurieren
Edit
Deffn Super(Mod%)=Gemdos(&H20,L:Mod%) ! 'Super' an/aus (s.GEMDOS(38))
```

Diese Prozedur beinhaltet die Hauptschleife, in welcher die Ereignisse verwaltet werden.

```
Procedure Main
  Repeat                              ! Hauptschleife
    Mouse X%,Y%,K%
    Key%=Asc(Right$(Inkey$))          ! Tastatur-Abfrage
    Bmove A%,Xbios(2)+2000,22000      ! 22000 Byte Speicherblock (A=Byte 1)
    '                                   ab Zeile 25 (80 Byte/Zeile =2000)
    '                                   in Bildschirmspeicher übertragen.
    If Key%=9                         ! Wenn ASCII=9 (<Tab>-Taste)
      If Romflg%=0                    ! Befinden wir uns im RAM?
        @Anzeige                      !   erstmal Anzeige aktualisieren,
        @Edit                         !   dann zur Editions-Routine.
      Else                            ! Wir sind im ROM!
        Get 0,0,639,24,Headline$      ! Infozeile des Hauptmodus sichern
        Deffill ,0,0
        Pbox 0,0,639,24               !----------.
        Deftext ,16,,13                           !
        Xt$="DAS ROM ('NUR-LESE-SPEICHER') KANN NICHT EDITIERT WERDEN !"
        Text 20,18,600,Xt$                        !
        Deffill ,2,4                              !-- Hinweis ausgeben
        Fill 60,2                                 !
        Pause 120                     !----------'
        Put 0,0,Headline$             ! Infozeile restaurieren
        Key%=0
      Endif
    Endif
    If Key%=98                        ! Helptaste gedrückt?
      Fix%=Fix% Xor 1                 ! dann im Wechsel die Byte-
    Endif                             !       Fixierung  an/aus
    If Key%=32                        ! <Space>-Taste gedrückt?
      Log%=Xbios(2)                   !   alte Logscreen
      Phys%=Xbios(3)                  !   alte Physscreen
      If A%<1840                      !   A%<1920 bedeutet, daß die Screen-
        '                             !   unter Null käme, wenn man den
        '                             !   Bildschirm-Speicher an den ge-
        '                             !   zeigten Ausschnitt anpaßt.
        B%=0                          !    deshalb Screenadresse auf Null
      Else                            !   sonst
        B%=A%-1840                    !    Screenadresse auf linke, obere
        '                             !    Ausschnittecke -2000 setzen,
        '                             !    damit der Screen-Versprung nicht
```

```
    Endif                          !              so stark ist.
    Void Xbios(5,L:B%,L:B%,-1)     !   Log- und Phys-Screen neu setzen
    Void Inp(2)                    !   auf Tastendruck warten
    Void Xbios(5,L:Log%,L:Phys%,-1) ! Screens wieder zurücksetzen
```

Dies ist der anfänglich beschriebene <SPACE>-Modus. Es kann ein Versatz von der BMOVE-Anzeige zu dieser Anzeige auftreten, da der Bmove-Befehl beliebige Startadressen annimmt, während der Bildschirmspeicher immer nur an Adressen gelegt werden kann, die durch 16 teilbar sind.

```
    Endif
    If Y%<25                       ! Wenn Mauszeiger über Anzeigefenster
      @U_scroll                    !   neuen Speicherbereich holen (-)
      Goto Marke                   !   und wieder von vorn
    Endif
    If Y%>299                      ! Wenn Mauszeiger unter Anzeige-
                                   ! fenster
      @D_scroll                    !   neuen Speicherbereich holen (+)
      Goto Marke                   !   und wieder von vorn
    Endif
    If K%                          ! Wenn eine Maustaste gedrückt wurde,
      @M_key                       !   dann zur Mausbehandlung
    Else                           ! keine Maustaste gedrückt!
      Dpoke 9952,Int(X%/8+0.5)*8   !   Zeiger auf Byteraster (8 Pixel)
    Endif                          !                     einstellen.
    If Y%>24 And Y%<300
      Box X%-1,Y%-1,X%+8,Y%+1      ! Mauszeigerersatz zeichnen
      Box X%-3,Y%-3,X%+10,Y%+3
    Endif
    Mouse Xx%,Yy%,K%               ! Neue Mausdaten
    If (Xx%=X% And Yy%=Y%) Or K%   ! Mausbewegung oder Knopf gedrückt?
      @Anzeige                     !   Wenn ja, Anzeige aktualisieren.
    Endif
    Marke:
  Until Key%=27                    ! Programmende nur mit <Esc>-Taste
Return
```

In der 'If'-Abfrage 'If Key%=98' in der letzten Prozedur habe ich einen ebenso simplen wie effektiven Trick eingesetzt. Die meisten von Ihnen würden die Flag-Umschaltung wahrscheinlich folgendermaßen gestalten:

```
If Key%=98 And Fix%=0
  Fix%=1
  Goto Label
Endif
If Key%=98 And Fix%=1
  Fix%=0
Endif
Label:
```

Durch 'Fix%=Fix% Xor 1' wird exakt dasselbe erreicht. Durch 'Xor' wird nämlich das Flag grundsätzlich auf den anderen Zustand (1 zu 0 / 0 zu 1) umgeschaltet.

```
Procedure Init                    ! Screen-Aufbau
  Cls
  Graphmode 2
  Deftext ,16,,13
  Text 122,15,135,"SPEICHER-SPION"
  Deffill ,0,0
  Deftext ,0,,
  Text 1,19," "
  Print At(3,1);A%''
  Graphmode 1
  Deftext ,0,,4
  Tx1$="<TAB>-Taste=Bit-Edit-Modus |"
  Tx1$=Tx1$+" Maustaste rechts/links=Byte +/- 1"
  Tx2$="Maus unter/über Bit-Fenster ="
  Tx2$=Tx2$+" Speicherscrolling (Byte +/- 160)"
  Tx3$="Zusätzlich Maustaste links/rechts/"
  Tx3$=Tx3$+"beide=Byte +/- 400/1600/3600"
  Text 265,6,Tx1$
  Text 265,13,Tx2$
  Text 265,20,Tx3$
  Print At(8,21);"Format:  Adresse:  Inhalt:"
  Print At(68,23);"Hexa:"
  Print At(68,21);"ASCII:"
  Print At(52,21);"<RAM>"
  Deffill ,2,4
  Pbox 470,308,525,394
  For I%=0 To 2
    Line 0,300+I%,639,300+I%
    Line 0,22+I%,639,22+I%
  Next I%
  Line 120,0,120,22
```

```
  Line 255,0,255,22
  Fill 200,1
  Box 479,315,512,348              ! Fenster für Longword-Bild
  Box 479,355,496,372              ! Fenster für Wort-Bild
  Box 479,380,488,389              ! Fenster für Byte-Bild
  Box 503,372,512,389              ! Fenster für Char-Bild
  Box 10,308,460,394
  Deffill ,1,1
Return
```

Soll der angezeigte Speicherbereich in Richtung Adresse Null verschoben werden, wird das hier erledigt.

```
Procedure U_scroll
  Showm
  If K% And A%>400*(K%^2)          ! Wenn Maustaste und neue BMOVE-
    '                              !                 Adresse > 0
    Sub A%,400*(K%^2)              !   dann, je nach Maustaste, um
    '                              !   400/1600/3600 Bytes vermindern
    If A%<1000 And Sc.flg%=1       !   wenn Zähler < 1000 und Umrech-
      '                            !   nungsflag gesetzt
      Sc.flg%=0                    !    Umrechnungsflag löschen
    Endif
    If A%>Ram_top%+10000 And A%<Rom_bot% ! Im leeren Bereich zwischen
      '                            !                     RAM und ROM?
      Romflg%=0                    !     auf RAM umschalten
      A%=Xbios(2)+(Int(((Ram_top%-10000)-Xbios(2))/160))*160
      '                            !     nächste Adresse so setzen, daß
      '                            !     Bildschirmspeicher ohne Offset
      '                            !     gescrollt wird.
      Print At(52,21);"<RAM>"
    Endif
  Else                             ! sonst
    Sub A%,160                     !   nur um 160 Byte vermindern
    If A%<1000 And Sc.flg%=1       !   wenn Zähler < 1000 und Umrech-
      '                            !   nungsflag gesetzt
      Sc.flg%=0                    !    Umrechnungsflag löschen
    Endif
    If A%<0                        !   Minus-Adressen gibt es nicht
      A%=0                         !    0 = kleinste lesbare Adresse
    Endif
    If A%>Ram_top%+10000 And A%<Rom_bot% ! Im leeren Bereich zwischen
      '                            !                     RAM und ROM?
```

```
      Romflg%=0                    !     auf RAM umschalten
      A%=Xbios(2)+(Int(((Ram_top%-10000)-Xbios(2))/160))*160
      '                            !     nächste Adresse wie oben
      Print At(52,21);"<RAM>"
    Endif
    @Anzeige                       !   und Anzeige aktualisieren
  Endif
  Print At(3,1);A%''               !   Adresse der linken, oberen
  '                                !       Ausschnittecke anzeigen
  Hidem
Return
```

Die Verschiebung des angezeigten Speicherbereichs in Richtung RAM-Ende findet hier statt.

```
Procedure D_scroll
  Showm
  If K% And A%<Rom_bot%+158000     ! Maustaste gedrückt und noch im
    '                              !                lesbaren Bereich?
    Add A%,400*(K%^2)              !   dann, je nach Maustaste, um
    '                              !   400/1600/3600 Bytes erhöhen
    If A%>1000 And Sc.flg%=0       !   Zähler > 1000 u.Umrechnungsflag=0
      Sc.flg%=1                    !   Umrechnungsflag=1
      A%=Xbios(2)-(Int((Xbios(2)-A%)/160)+1)*160
      '                            !    Zähler so setzen, daß der Bild-
      '                            !    schirmspeicher ohne störendes
      '                            !    Offset gescrollt wird
    Endif
    If A%>Ram_top% And A%<Rom_bot%-10000 ! Im leeren Bereich zwischen
      '                            !                          RAM und ROM?
      Romflg%=1                    !    dann auf ROM umschalten
      A%=Rom_bot%-10000            !    Zähler auf 10000 Byte unter
      Print At(52,21);">ROM<"      !                ROM-Start setzen
    Endif
  Else                             ! keine Maustaste gedrückt, bzw.
    '                              !           oberhalb des ROM-Endes
    Add A%,160                     !   nur 160 Byte (2 Zeilen/Schritt)
    If A%>1000 And Sc.flg%=0       !   Zähler > 1000 u.Umrechnungsflag=0
      Sc.flg%=1                    !   Umrechnungsflag=1
      A%=Xbios(2)-(Int((Xbios(2)-A%)/160)+1)*160
      '                            !    Zähler so setzen, daß der Bild-
      '                            !    schirmspeicher ohne störendes
      '                            !    Offset gescrollt wird
    Endif
```

```
    If A%>Rom_bot%+158000          !  oberhalb ROM-Ende?
      Sub A%,160                   !  Zähler-Addition rückgängig machen
    Endif
    If A%>Ram_top% And A%<Rom_bot%-10000 ! Im leeren Bereich zwischen
      '                            !                    RAM und ROM?
      Romflg%=1                    !   dann auf ROM umschalten
      A%=Rom_bot%-10000            !   Zähler auf 10000 Byte unter
      Print At(52,21);">ROM<"      !              ROM-Start setzen
    Endif
    @Anzeige                       !  und Anzeige aktualisieren
  Endif
  Print At(3,1);A%''               !  Adresse der linken, oberen
  '                                !      Ausschnittecke anzeigen
  Hidem
Return
```

In den beiden vorangegangenen Prozeduren werden Sie sich vielleicht über das Umschalten von RAM auf ROM und zurück gewundert haben. Das heißt nichts anderes, als daß diese beiden Speicher um etliche MEGA-Adressen auseinander liegen und eine 2Bomben-Errormeldung dadurch verhindert wird, daß der dazwischen liegende Bereich, der für den BMOVE-Befehl nicht erreichbar ist, übersprungen wird. Innerhalb des ROMs kann der BMOVE-Befehl dann wieder eingesetzt werden. Allerdings nur in Richtung RAM.

Hier werden die Maustasten-Funktionen im Haupt-Modus ausgeführt.

```
Procedure M_key
  If K%=1                          ! Wenn linke Maustaste
    If X%>7                        !   und X-Koordinate größer 7
      Dpoke 9952,Int(X%/8)*8-8     !     dann Mauszeiger im Byteraster
      '                                  um 1 Byte nach links und
      Goto Weiter                  !     nächste Abfrage überspringen
    Endif
    If X%=<7 And Y%>24             !   Zeiger auf erstem Byte der Zeile?
      Dpoke 9954,Y%-1              !     dann eine Zeile hoch
      Dpoke 9952,639               !     und auf letztes Byte setzen
    Endif
  Endif
```

```
  If K%=2                          ! Wenn rechte Maustaste
    If X%<631                      !   und X-Koordinate kleiner 631
      Dpoke 9952,Int(X%/8)*8+8     !     dann Mauszeiger im Byteraster
      '                                  um 1 Byte nach rechts und
      Goto Weiter                  !     nächste Abfrage überspringen
    Endif
    If X%=>631 And Y%<300          !   Zeiger auf letztem Byte der
Zeile?
      Dpoke 9954,Y%+1              !     Dann eine Zeile abwärts
      Dpoke 9952,0
    Endif
    Weiter:
  Endif
Return
```

Die verwendeten DPOKEs in der letzten Prozedur werden am Ende des Kapitels unter 'Spezial-Adressen' beschrieben.

Durch die Routine M-key wird bewirkt, daß nur durch acht teilbare Positionen auf dem Bildschirm in X-Richtung eingenommen werden können, da die Bytes ja bekanntlicherweise aus 8 Bits (= 8 Pixel) bestehen. Der Witz an der Sache ist, daß mit diesen Dpokes der Original-Mauszeiger (der in diesem Fall ausgeschaltet ist) mitpositioniert wird. Es findet also eine echte Maus-Skalierung statt!

Unter dem großen Bit-Fenster befindet sich das Anzeigefenster für die aktuellen Byte/Word/Longword-Inhalte, sowie die ASCII- und Hexa-Anzeige, das Bitmuster des aktuellen Longwords und die Longword-, Word-, Byte- und Char-Images. Diese Anzeige spielt sich in der folgenden Prozedur ab.

```
Procedure Anzeige
  If Fix%=0                        ! Byte-Fixierung aus?
    Byte%=A%+(Y%-25)*80+Int(X%/8)  ! zu untersuchende Byteadresse:
  Endif
  If Byte%<1                       !
    Byte%=1
  Endif
  '                                A% = erstes Byte links oben im
  '                                  Fenster
```

```
  '                                  + ((MausY-Rand oben)*80 Byte je
  '                                    Zeile)
  '                                  + Integer(MausX / 8 Pixel je Byte)
  Word%=Int(Byte%/2)*2             ! zugehöriges Wort feststellen:
  '                                  (Integer(Byteadresse / 2)) * 2
  '                                   = gerade Wortadresse
  Bin_Puffer$=String$(32,"0")      ! Puffer für Binärzahl vorbereiten
  Rset Bin_Puffer$=Bin$(Lpeek(Word%)) ! Binär-Langwort einsetzen
  For I%=1 To 4                    ! 4 Hexa-Bytestrings
    Hex_buff$(I%)=String$(2,"0")   ! jeweils mit zwei Nullen vorbereiten
    Rset Hex_buff$(I%)=Hex$(Peek(Word%+I%-1)) ! und jeweils das Hexa-
    '                                           Byte einsetzen (4 Byte
    '                                           ab Wortadresse)
  Next I%                          ! nächstes Byte.
  '                                ! Und Werte anzeigen:
  Print At(8,22);"1 LONG ab "'Word%'''''Lpeek(Word%)''''''''''''
  Print At(8,23);"2 WORD ab
"'Word%'''''Dpeek(Word%)'Dpeek(Word%+2)''''''''
  Print At(8,24);"4 BYTE ab "'Word%'''''
  Print Peek(Word%)'Peek(Word%+1)'Peek(Word%+2)'Peek(Word%+3)''''''''
  Print At(68,22)'
  For I%=0 To 3
    Out 5,Peek(Word%+I%)           ! Alle 4 Byte als ASCII-Zeichen zeigen
  Next I%
  Print ''''
  Print At(68,24);Hex_buff$(1)'Hex_buff$(2)'Hex_buff$(3)'Hex_buff$(4)'
  For I%=0 To 127 Step 4                          !32 Langworte im
    Lpoke Xbios(2)+25340+(I%/4)*80,Lpeek(Word%+I%)! Langwort-Fenster
    '                                               zeigen.
    If Mid$(Bin_Puffer$,Int(I%/4)+1,1)="1"        ! Wenn Bit
    '                                             ! "Int(I%/4)+1"
    '                                             ! gesetzt ist,
      Deffill ,1,1                                ! Füllmuster auf
    '                                             ! schwarz
      Pbox 12+I%/4*14,385,24+I%/4*14,392          ! und Bit groß
    '                                             !  zeichnen.
    Else                                          ! Wenn nicht gesetzt,
      Deffill ,0,0                                ! Füllmuster auf weiß
      Pbox 12+I%/4*14,385,24+I%/4*14,392          ! u. groß gezeichnetes
    Endif                                         ! Bit löschen.
  Next I%
  For I%=0 To 31 Step 2                           ! 16 Worte im
    Dpoke Xbios(2)+28540+(I%/2)*80,Dpeek(Word%+I%)! Wort-Fenster
    '                                             ! anzeigen.
  Next I%
```

```
  For I%=0 To 7                                  ! 8 Byte im
    Poke Xbios(2)+30540+I%*80,Peek(Byte%+I%)     ! Byte-Fenster
    '                                            ! anzeigen.
  Next I%
  For I%=0 To 15                                 ! 16 Byte im
    Poke Xbios(2)+29903+I%*80,Peek(Byte%+I%*256) ! Char-Fenster anzeigen
  Next I%
Return
```

Die letzte Schleife 'Char-Fenster' ist eigentlich nur für die interessant, die noch mit einem Disketten-TOS arbeiten. Da das von Diskette geladene TOS in den ersten 200 KByte des Speichers abgelegt wird, hat man hier direkten Zugriff auf den Zeichensatz-Speicher (s. Spezial-Adressen).

Ein Speicher-Spion ohne die Möglichkeit, in das Geschehen eingreifen zu können, wäre nur die Hälfte wert. Mit der nächsten Prozedur wird die Edition des Speicherinhalts ermöglicht.

```
Procedure Edit
  Get 0,0,639,24,Headline$            ! Infozeile des Hauptmodus sichern
  Fix%=0                              ! Byte-Fixierung ausschalten
  Showm
  E.start3:
  Deffill ,0,0                                                  !--|
  Pbox 0,0,639,24                                               !  |
  Deftext ,16,,13                                               !  |
  Text 10,18,"EDIT-MODUS"                                       !  |
  Line 123,0,123,23                                             !  |
  Deffill ,2,4                                                  !  |Aufbau
  Fill 60,2                                                     !  |der
  Deftext ,0,,4                                                 !  |Info-
  Tx1$=" <Tab> = Image load / save| <Esc>  =   Edit - Ende   |"!  |Zeile
  Tx1$=Tx1$+" Das Longword-Bitmuster unter"                     !  |für
  Tx2$=" <-Pfeil-Taste = Byte-1 | >-Pfeil-Taste = Byte+1     |"!  |den
  Tx2$=Tx2$+" der Ziffern-Anzeige kann mit"                     !  |Edit-
  Tx3$=" ^-Pfeil_Taste = Byte-80 | v-Pfeil-Taste = Byte+80   |"!  |Modus
  Tx3$=Tx3$+" der Maus editiert werden."                        !  |
  Text 130,7,Tx1$                                               !  |
  Text 130,14,Tx2$                                              !  |
  Text 130,21,Tx3$                                              !--|
  Repeat                          ! Editor-Schleife
    Mouse Ex%,Ey%,K%              ! Neue Mauswerte
```

```
If Ey%<300                        ! Nur wenn Mauszeiger im Blockfenster
  Bmove A%,Xbios(2)+2000,22000 !   22000 Byte Speicherblock anzeigen
  Box X%-1,Y%-1,X%+8,Y%+1         ! Mauszeigerersatz zeichnen
  Box X%-3,Y%-3,X%+10,Y%+3
Endif
R.key%=Asc(Right$(Inkey$))        ! Tastatur abfragen`
Repeat
Until Len(Inkey$)=0               ! Tastaturpuffer leeren
If K%=2                           ! rechte Maustaste gedrückt?
  @Snap                           !  zur Schnappschuß-Prozedur
Endif
If K%=1 And Ex%>13 And Ex%<458 And Ey%>384 And Ey%<393!Maustaste
  '                               ! gedrückt und Mauszeiger auf
  '                               ! Editorfeld?
  Index%=Int((Ex%-12)/14)         ! Index des gewählten Bits ermitteln
  I.bit%=(Point(19+Index%*14,388)) Xor 1  ! gewähltes Bit umschalten
  Deffill ,I.bit%,I.bit%
  Pbox 12+Index%*14,385,24+Index%*14,392        ! Bit zeichnen
  If Index%=0                                   ! Wenn Negativ-Bit
    '                                           ! (32.) gewählt,
    A2%=Lpeek(Word%) Xor (2^(31-Index%)*(-1)) !   dann XOR negiert.
  Else                                          ! Bits 1-31 gewählt!
    A2%=Lpeek(Word%) Xor (2^(31-Index%))        !   Dann Bit umkehren
  Endif
```

Diese beiden merkwürdigen If-Abfragen haben den Grund, daß im größten, vom ST verdaubaren Zahlenformat (Bit 0 - 31 = 32 Bit) das letzte Bit (Bit 31) zur Identifikation des Vorzeichens verwendet wird. Da wir aber volle 32 Bits analysieren wollen, muß die automatische Umschaltung von Positiv- auf Negativwerte - sobald das Bit 31 gesetzt ist - unterdrückt werden. Genau das wird durch die XOR-Negierung hier erreicht.

```
  Lpoke Word%,A2%                         ! In Speicher zurück
  Bmove A%,Xbios(2)+2000,22000            ! Speicherblock anzeigen
  Box X%-1,Y%-1,X%+8,Y%+1                 ! Mauszeigerersatz zeichnen
  Box X%-3,Y%-3,X%+10,Y%+3
  @Anzeige                                ! Anzeige aktualisieren
  Pause 5
Endif
If R.key%                                 ! Tastatur?
  If R.key%=9                             ! Tab-Taste ?
    Alert 2,"Image laden oder speichern?",1,"LOAD|SAVE",I.bk%
```

```
        If I.bk%=2                          ! Image speichern?
          Bmove A%,Xbios(2)+2000,22000      !  Speicherblock anzeigen
          Sget F.scr$                       !  Screen sichern
          @Imagesave                        !  Image speichern
        Else                                ! Image laden!
          @Imageload(Word%)                 !  dann lade
        Endif
      Endif
      If R.key%=77                  ! Rechts-Pfeiltaste gedrückt?
        Add X%,8                    !   X-Koordinate 1 Byte nach rechts,
        K%=2                        !   rechte Maustaste simulieren,
        @M_key                      !   Speicherscrolling-Routine ( + )
      Endif
      If R.key%=75                  ! Links-Pfeiltaste gedrückt?
        Sub X%,8                    !   X-Koordinate 1 Byte nach links,
        K%=1                        !   linke Maustaste simulieren,
        @M_key                      !   Speicherscrolling-Routine ( - )
      Endif
      If R.key%=80 And Y%<299       ! Abwärts-Pfeiltaste gedrückt?
        Add Y%,1                    !   Y-Koordinate 1 Zeile abwärts
      Endif
      If R.key%=72 And Y%>27        ! Aufwärts-Pfeiltaste gedrückt?
        Sub Y%,1                    !   Y-Koordinate 1 Zeile aufwärts
      Endif
      Bmove A%,Xbios(2)+2000,22000  ! Speicherblock anzeigen
      @Anzeige                      ! und Anzeige aktualisieren
    Endif
  Until R.key%=27                 ! Schleife verlassen, wenn <Esc>-Taste
  Put 0,0,Headline$               ! Infozeile für Hauptmodus restaurieren
  Hidem
Return
```

In der nächsten Prozedur werden beliebige Word- bzw. Longword-Images (16 bzw. 32 Bits breit) im Dezimal- und Binärforat als Datazeilen auf Diskette abgespeichert. Die Binärfiles erhalten die Extension 'BI', sowie entweder den Wert 1 (=Word-Image) oder 2 (=Longword-Image). Dies ist notwenig, um die verschiedenen Formate auf Diskette unterscheiden zu können. Wollen Sie also ein Word-Binär-Image laden, wählen Sie eine Datei mit der Extension '.BI1'. Die Dezimal-Files tragen die Extension 'DZ' sowie die Identifikationsziffer. Diese Data-Files lassen sich über die Interpretermenü-Funktion 'Merge' in den Programmspeicher laden. Vorher muß jedoch die Extension '.LST' in der Indexzeile

der 'Merge'- Fileselectbox gelöscht und der graue Kopfbalken angeklickt werden. Die Binär-Image-Dateien lassen sich mit dem Programm 'IMAGEDIT.BAS' laden und weiterverarbeiten.

Dabei ist zu beachten, daß die hier gespeicherten Word-Dezimal-Images noch keinen Mausaktionspunkt beinhalten. Wollen Sie diese Images für Mauszeiger in 'IMAGLOAD.BAS' verwenden, müssen Sie vorher die Binär-Files mit 'IMAGEDIT.BAS' laden und wieder abspeichern. Anschließend können Sie den Aktionspunkt bestimmen. Die dazugehörige Word-Dezimal-Image-Datei wird dann um diese beiden Werte (X/Y-Rasterkoord. des Aktionspunktes) erweitert.

```
Procedure Imagesave
  Local I.bk2%,Fl$,Num_fl$,L$,D.peek$,L.peek$,El$,I%,J%
  Alert 2,"16Bit oder 32Bit-Image?",2,"16Bit|32Bit",I.bk2%
  Al$="Aus dem Image-Fenster|oder aus der großen|Bit-Screen?"
  Alert 2,Al$,1,"IMAGE|SCREEN",I.bk3%
  Wrdmem%=Word%                        ! aktuell angezeigtes Word sichern
  If I.bk3%=2                          !  Image aus der Screen holen?
    Graphmode 3
    Sput F.scr$                        !  Screen restaurieren
    Repeat                             !  Positionierungs-Schleife
      Mouse Xi%,Yi%,K%                 !   Mausstatus
      Box Xi%-16*I.bk2%-1,Yi%-16*I.bk2%-1,Xi%+2,Yi%+2 !Auswahl-Rahmen
      Box Xi%-16*I.bk2%,Yi%-16*I.bk2%,Xi%+1,Yi%+1     !   "      "
      Poke 3583,0                      !   Maus-Links/Rechts-Bewegung aus
      '                                !      (s. 'Spezial-Adressen')
      Poke 3584,0                      !   Maus-Auf/Ab-Bewegung aus
      Repeat                           !   Warteschleife
      Until Mousek Or Peek(3583) Or Peek(3584)! bis Maustaste gedrückt
      '                                !   oder Maus bewegt wurde
      Box Xi%-16*I.bk2%-1,Yi%-16*I.bk2%-1,Xi%+2,Yi%+2 ! Rahmen aus
      Box Xi%-16*I.bk2%,Yi%-16*I.bk2%,Xi%+1,Yi%+1     !   "    "
    Until Mousek                       !  Exit, wenn Maustaste
    Get Xi%-16*I.bk2%+1,Yi%-16*I.bk2%+1,Xi%,Yi%,Img$ ! Image puffern
    Graphmode 1
    Word%=Varptr(Img$)+6               ! Lese-Zeiger auf Puffer-Anfang
    If Mousek=2                        ! Rechte Maustaste gedrückt?
      Goto No.load                     !  Abbruch
    Endif
  Endif
  Fileselect "\*.BI"+Str$(I.bk2%),".BI"+Str$(I.bk2%),Fl$
  '                                    ! SAVE-Datei wählen
```

```
If Fl$="" Or Fl$="\.BI"+Str$(I.bk2%)! ungültiger Dateiname?
  Goto No.load                        !  dann Abbruch
Endif
If Instr(Right$(Fl$,4),".")=0         ! keine Extension gesetzt?
  Fl$=Fl$+".BI"+Str$(I.bk2%)          !  dann nachholen
Endif
If Right$(Fl$,3)<>"BI"+Str$(I.bk2%)!  falsche Extension?
  Mid$(Fl$,Len(Fl$)-3,3)="BI"+Str$(I.bk2%)! dann berichtigen
Endif
Num_fl$=Left$(Fl$,Len(Fl$)-3)+"DZ"+Str$(I.bk2%) ! Dezimal-Filename
'                                               ! bilden
Open "O",#98,Num_fl$                  ! Dezimal-File öffnen
Open "O",#99,Fl$                      ! Binär-File öffnen
For I%=0 To I.bk2%*16-1               ! alle Imagezeilen durchgehen
  Clr El$                             !  1.Puffer löschen
  L$=String$(16*I.bk2%,"0")           !  2.Puffer vorbereiten
  If I.bk2%=1                         !  Word-Image speichern?
    D.peek$=Bin$(Dpeek(Word%+I%*2))!   Binärstring bilden
    Mid$(L$,16-Len(D.peek$)+1,Len(D.peek$))=D.peek$ ! in 2.Puffer
  '                                                 ! einbauen
  Else                                ! Longword-Image speichern!
    L.peek$=Bin$(Lpeek(Word%+I%*4))!   Binärstring bilden
    Mid$(L$,32-Len(L.peek$)+1,Len(L.peek$))=L.peek$ ! in 2.Puffer
  '                                                 ! einbauen
  Endif
  For J%=0 To I.bk2%*16-1             ! alle Bits der Zeile durchgehen
    If Mid$(L$,J%+1,1)="1"            !  Bit gesetzt?
      El$=El$+"1"                     !   '1' in 1. Puffer eintragen
    Else                              !  Bit nicht gesetzt!
      El$=El$+"0"                     !   '0' in 1. Puffer eintragen
    Endif
  Next J%                             ! nächstes Bit
```

Hier sehen Sie, wozu die 'Bin$'-Funktion verwendet werden kann. Hätten wir diese Funktion nicht zur Verfügung, müßte man die gesetzten Bits in einem Word nach folgendem Schema ermitteln:

```
Xyz%=25536
For N.bit%=15 Downto 0
  If Xyz% And 2^N.bit%
    X$=X$+"1"
```

```
      Else
        X$=X$+"0"
      Endif
    Next N.bit%
    X%=Val("&X"+X$)
    Print Bin$(Xyz%)
    Print Bin$(X%)
```

Bei einem Word ist das auch noch recht einfach. Bei einem Longword wird dieses Verfahren schon etwas komplizierter. Deshalb setze ich oben im Programm einfach einen Binärstring in einen vorbereiteten Stringpuffer und frage der Reihe nach die Zeichen des Strings ab.

```
    If I% Mod 16=0                  !   erste Data-Zeilenposition?
      Print #98;"D ";               !    'D' (Data) in File schreiben
    Endif
    Print #98;Val("&X"+El$);        !   Dezimal-Zeilenwert in File
schreiben
    If (I%+1) Mod 16=0              !   Data-Zeile (16 Werte) voll?
      Print #98                     !    Zeilenende schreiben
    Else                            !   noch nicht voll!
      Print #98;",";                !    Data-Trennkomma schreiben
    Endif
    Print #99;"D ";El$              !   Binärzeile schreiben
  Next I%
  Close #98                         ! Dezimal-Datei schließen
  Close #99                         ! Binär-Datei schließen
  No.load:
  Word%=Wrdmem%                     ! Original-Anzeige-Word restaurieren
Return
```

Mit dieser Prozedur lassen sich Disketten-Imagefiles (s. unter 'Image-Loader') in den Speicher übertragen. Sie werden ab der aktuellen Word-Position eingesetzt.

```
Procedure Imageload(Adr%)
  Local Trns$,I%,Pkt%,Frm%,P.line$,P_name$
  Fileselect "\*.BI?",".BI",P_name$   !  'BI'-Datei wählen
  If Exist(P_name$) And Left$(Right$(P_name$,3),2)="BI"
    '                                 ! Datei vorhanden?
    Open "I",#1,P_name$               !    Datei öffnen
```

```
    Frm%=Val(Right$(P_name$))*16        !    Format feststellen (16 / 32)
    For I%=1 To Frm%                    !      Lese-Schleife
      Input #1,P.line$                  !        Bitmusterzeile lesen
      Trns$=Trns$+Right$(P.line$,Frm%)!          Gesamtstring bilden
    Next I%
    Close #1
    For Pkt%=0 To Frm%-1                !    Ausgabe-Schleife
      B.wert%=Val("&X"+Mid$(Trns$,Pkt%*Frm%+1,Frm%)) ! Dez.-Wert bilden
      If Frm%=32                        !      Longword-Muster ?
        Lpoke Adr%+Pkt%*4,B.wert%       !        dann Wert L-Poken
      Endif
      If Frm%=16                        !      Word-Muster ?
        Dpoke Adr%+Pkt%*2,B.wert%       !        dann Wert D-Poken
      Endif
    Next Pkt%                           !    nächste Imagezeile
  Endif
Return
```

Wer schon einmal mit 'WORDPLUS' gearbeitet hat, wird evtl. den Nutzen von 'SNAPSHOT.ACC' kennengelernt haben. Diese Prozedur speichert ebenfalls beliebige Bildausschnitte im GFA-GET/PUT-Format ab, um sie hinterher anderweitig weiterverwenden zu können. Im Programm 'GFA_DMON.BAS' finden Sie eine Prozedur 'Snapload', die Sie zum Laden und Plazieren der Ausschnitte verwenden können. Am Ende dieses Kapitels finden Sie unter 'Aus Data mach' PUT' auch den Aufbau dieses Spezialstrings erläutert.

```
Procedure Snap
  Local Snap$,Xx%,Yy%,X%,Y%,K%,Sn.bk%,Bz.bk%
  Alert 2,"Snappschuß speichern?",1,"OKAY|NEIN",Sn.bk%
  If Sn.bk%=1                        ! Schnappschuß speichern?
    Alert 2,"Byte-Zeiger an oder aus?",1," AN | AUS ",Bz.bk%
    If Bz.bk%=2                      ! Bytezeiger löschen?
      Bmove A%,Xbios(2)+2000,22000   !  dann Speicherblock neu
    Endif
    Defmouse 3                       ! Zeigefinger-Maus
    Repeat
    Until Mousek                     ! Auf Mausklick warten
    If Mousek=1                      ! linke Maustaste gedrückt?
      Graphmode 3
      Mouse Xx%,Yy%,K%               !   Mauskoord. linke/obere Ecke
```

```
      Repeat                          !   Gummi-Rahmen-Schleife
        Mouse X%,Y%,K%                !     Mauskoord. rechte/untere Ecke
        Box Xx%,Yy%,X%,Y%             !     Box zeichnen
        Poke 3583,0                   !     Maus-Links/Rechts-Bewegung aus
        '                             !     s. 'Spezial-Adressen'
        Poke 3584,0                   !     Maus-Auf/Ab-Bewegung aus
        Repeat                        !     Warteschleife
        Until Mousek=0 Or Peek(3583)  Or Peek(3584)! bis Maustaste frei
        '                             !     oder Maus bewegt wurde
        Box Xx%,Yy%,X%,Y%             !     Box löschen
      Until Mousek=0                  !     Exit, wenn Maustaste frei
      Get Xx%,Yy%+19,X%,Y%+19,Snap$ !   Ausschnitt puffern
      Fileselect "A:\*.*",".SNP",S$ !   Snap-Datei auswählen
      Deftext ,0,,6
      If S$>"" And S$<>"A:\.SNP"    !   gültiger Dateiname?
        If Instr(Right$(S$,4),".")=0!    keine Extension gesetzt?
          S$=S$+".SNP"                !     dann nachholen
        Endif
        If Right$(S$,3)<>"SNP"        !   falsche Extension?
          Mid$(S$,Len(S$)-3,3)="SNP"!       dann berichtigen
        Endif
        Bsave S$,Varptr(Snap$),Len(Snap$) ! Snapfile speichern
      Endif
    Endif
    Graphmode 1
    Defmouse 0
  Endif
  Pause 10
Return
```

1.2 Tischlein deck' dich

Auf der Diskette finden Sie einen Ordner mit Namen 'IMAGES'. Er enthält schon einen komplett zusammengestellten Austauschsatz für alle AES-Images. Um zu sehen, wie ein eigenes Desktop, eigene Alerts und eigene Mäuse aussehen könnten, starten Sie einfach das Programm 'IMAGLOAD.BAS', öffnen bei jeder Abfrage diesen Ordner und klicken anschließend 'OK' an. Verlassen Sie nun das GFA-BASIC, und schauen Sie sich das Desktop an.

Übrigens, wundern Sie sich nicht darüber, daß ich immer wieder in systemloser Unregelmäßigkeit einmal 'Image' und einmal 'Icon' schreibe. Es ist völlig egal, wie Sie die kleinen Bildchen nennen, beide Ausdrücke heißen im übertragenen Sinn ganz einfach 'Bild'.

Wurden mit dem Programm 'IMAGEDIT.BAS' (s. unter 'Image-Editor') Desktop-, Alert- oder Maus-Images entwickelt oder mit 'MEMO_SPY.BAS' solche Icons abgespeichert, können diese hiermit in den Image-Speicher des AES eingebaut werden. Maus-Icons müssen jedoch vorher grundsätzlich mit 'IMAGEDIT.BAS' abgespeichert werden, da nur dort die jeweiligen Aktionspunkt-Koordinaten in das Imagefile eingebaut werden. Außerdem müssen für alle Desktop- und Maus-Icons die dazugehörigen Masken-Files vorhanden sein. Für diesen Loader werden nur die Dezimal-Files der Icons benötigt. Die Binär-Files sind hauptsächlich zur Weiterverarbeitung in eigenen Programmen gedacht.

Um die Dateien hiermit ohne großen Aufwand laden zu können, sind sie unter feststehenden Dateinamen abzuspeichern. Wollen Sie mehrere Austauschsätze vorrätig halten, müssen Sie sie unter diesen Namen in verschiedenen Ordnern abspeichern. Obwohl Sie hier nach einem Ordner gefragt werden, können die Icons auch auf der obersten Directory-Ebene gespeichert sein, solange alle zu einem Satz gehörigen Dateien mit diesen festgelegten Dateinamen ebenfalls auf dieser Ebene liegen.

Die Maus-Icon-Dateien sind mit dem Namen 'MAUSICN?.DZ1' abzuspeichern. Für die Mausmasken-Dateien gilt der Name 'MAUSMSK?.DZ1'.

Das Fragezeichen steht hier für eine Ziffer, die von Ihnen angegeben wird. Mit dieser Ziffer wird bestimmt, an welche 'Defmouse'-Position der neue Mauszeiger gesetzt werden soll. Also der Mauszeiger mit der Bezeichnung 'MAUSICN1.DZ1' und 'MAUSMSK1.DZ1' kann dann später mit dem Befehl 'DEFMOUSE 1' aufgerufen werden. Die Ziffern sind durch die Defmouse-Parameter natürlich auf den Bereich 0-7 begrenzt.

Für die Desktop-Icons gilt 'DESKICN?.DZ2' und für die entsprechenden Desktop-Masken 'DESKMSK?.DZ2'.

Hier steht das Fragezeichen für eine Ziffer im Bereich von 1 bis 5. Es können also nur 5 Icons ausgetauscht werden:

1. Laufwerk-Symbol
2. Ordner-Symbol
3. Mülleimer-Symbol
4. Programm-Symbol
5. Datei-Symbol

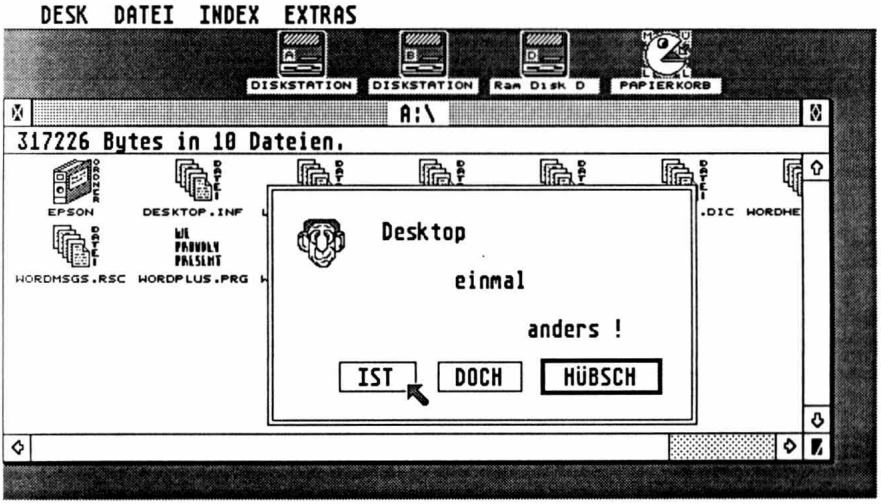


Abb. 2: **Desktop einmal anders**

Die Alertbox-Icons tragen den Namen 'ALRTICN?.DZ2'. Anstatt des Fragezeichens können hier die Ziffern 1 bis 3 verwendet werden. Es gibt nur die Möglichkeit, 3 verschiedene Icons als Verzierung für Alertboxen gleichzeitig zu verwalten.

ALRTICN1.DZ2 kann später durch 'Alert 1,.....',
ALRTICN2.DZ2 kann später durch 'Alert 2,.....' und
ALRTICN3.DZ2 kann später durch 'Alert 3,.....'

aufgerufen werden. Mit den Prozeduren im Programm 'SET_ALRT.BAS' können diese Icons auch während eines eigenen Programms beliebig oft ausgetauscht werden, um so grafisch die verschiedenen Alertboxen untermalen zu können.

Bei den Alert-Icons wird keine Maske benötigt, da sie grundsätzlich auf weißem (bzw. hintergrundfarbenem) Untergrund angezeigt werden.

```
' Programm: IMAGLOAD.BAS
'
' |""""""""""""""""""""""""""""""""""""""""""""|
' |            IMAGE-LOADER                    |
' |--------------------------------------------|
' |""""""""""""""""""""""""""""""""""""""""""""|

Print At(18,4);"*************************************************"
Print At(18,5);"*                                               *"
Print At(18,6);"*  Was soll ausgetauscht werden ?                *"
Print At(18,7);"*  ------------------------------------------   *"
Print At(18,8);"*                                               *"
Print At(18,9);"*  Maussatz                                (1)  *"
Print At(18,10);"*  Desksatz                                (2)  *"
Print At(18,11);"*  Alertsatz                               (3)  *"
Print At(18,12);"*  Desksatz  +  Maussatz                   (4)  *"
Print At(18,13);"*  Maussatz  +  Alertsatz                  (5)  *"
Print At(18,14);"*  Alertsatz +  Desksatz                   (6)  *"
Print At(18,15);"*  Maussatz  +  Desksatz + Alertsatz       (7)  *"
Print At(18,16);"*  Abbruch                                 (8)  *"
Print At(18,17);"*                                              *"
Print At(18,18);"*************************************************"
Repeat                              ! Eingabeschleife
  A%=Val(Input$(1))                 !  Menüpunkt eingeben
  If A%=8                           !  Abbruch?
    Edit                            !   Abbruch!
  Endif
Until A%>0 And A%<8                 !  gültige Ziffer eingegeben?
Cls
On A% Gosub P1,P2,P3,P4,P5,P6,P7    !  Verteiler
```

Mit der Prozedur 'P1' wird der komplette Maus-Icon- und Maskensatz ausgewechselt.

```
Procedure P1
  A$=""
  Restore Maus_0_icn                  ! Data-Zeiger setzen
  For I%=0 To 15                      ! 16 Maus-Icon-Zeilen
    Read Dat%                         !  lesen
    A$=A$+Mki$(Dat%)                  !  String bilden
  Next I%
  B$=Left$(A$,Len(A$)-1)              ! A$ auseinandernehmen, damit
  C$=Right$(A$)                       ! sich der String nicht im
  '                                   ! Speicher selber findet.
  A$=Space$(Len(A$))                  ! A$ wieder löschen
```

Der hier in A$ gebildete String ist ja auch im Speicher vorhanden. Er soll sich aber nicht selbst finden, sondern andere Strings, die denselben Inhalt haben. Deswegen wird hier der String bis auf das äußerste rechte Zeichen reduziert. Das letzte Zeichen des Strings wird separat gespeichert. Diese beiden Teilstrings werden an die Such-Prozedur übergeben und dort wieder zusammengesetzt. Vorher wird allerdings der Originalstring wieder gelöscht. Trotzdem wird sich der String mindestens einmal selber finden. Und zwar an der Stelle im Speicher, die der Interpreter benutzt, um bei der Prozedur-Abarbeitung den Suchstring zusammenzusetzen. Würde man jedoch nur einen String übergeben, würde er sich ständig selbst finden, da er durch die Garbage-Collection immer weiter kopiert werden würde, bzw. weil er ja dann auch im Arbeitsspeicher steht.

Die drei numerischen Parameter bedeuten hier die Start- und die Endadresse des zu durchsuchenden Speicherbereichs, sowie ein Flag, das in der Suchprozedur bewirkt, daß der richtige Offset vom gesuchten Image zum entsprechenden Blockanfang zurückgerechnet wird.

```
  @Find(B$,C$,2048,250000,1,*Adr%)  ! String im Speicher suchen
  If Adr%                           ! String gefunden?
    Print At(25,3);"Mausordner öffnen und OK klicken"
    Fileselect "\*.BI1","DEFMOUSE",Maus$   ! Ordner öffnen
    Cls
    For I%=0 To 7                          ! 8 Maus-Icons
      Open "I",#1,Maus$+"MAUSMSK"+Str$(I%)+".DZ1"!Icondatei öffnen
```

```
      Seek #1,2                          ! Filepointer auf drittes
      '                                  ! Byte setzen (Byte 1 u. 2
      '                                  ! enthalten 'D ' = Abk. für
      '                                  ! 'Data ').
      For J%=0 To 15                     ! 16 Iconzeilen
        Input #1;A%                      !   Zeilenwert lesen
        Dpoke Adr%+I%*74+J%*2,A%         !   in AES-Speicher
        '                                !   eintragen
      Next J%                            ! nächste Zeile
      Close #1                           ! Datei schließen
      Open "I",#1,Maus$+"MAUSICN"+Str$(I%)+".DZ1"!Maskendatei öffnen
      Seek #1,2                          ! siehe oben
      For J%=0 To 15                     ! 16 Maskenzeilen
        Input #1;A%                      !  Zeilenwert lesen
        Dpoke Adr%+32+I%*74+J%*2,A%      !  in Speicher eintragen
      Next J%                            ! nächste Zeile
      Input #1;A%;B%                     ! Aktionspunkt-X,-Y lesen
      Lpoke Adr%-10+I%*74,A%*65536+B%    ! in Speicher schreiben
      Close #1                           ! Maskendatei schließen
    Next I%                              ! Nächstes Icon
  Endif
  Maus_0_icn:
  Data 2048,2108,98,1730,50820,6538,6996,1760,7512,13308,24928,17118,
17624,19030,13332,0,
Return
```

Mit der ersten For/Next-Schleife in der obigen Prozedur und dem folgenden '@Find'-Aufruf wird zuerst die Data-Zeile unter 'Maus_0_icn:' eingelesen und im Speicher nach der Adresse ihres Vorkommens gesucht.

Da nicht immer 100%ig feststeht, wo sich die AES-Icons befinden, muß auf diese Art ihre Position festgestellt werden, um die neuen Icons an die richtige Adresse schreiben zu können. Der Grund dafür, warum die Icons an unterschiedlichen Stellen liegen können, ist hauptsächlich darin zu sehen, daß mit dem 'Auto'-Ordner gebootete Programme die Adressenlage verschieben können.

Allen Befürchtungen zum Trotz, daß evtl. die neuen MEGA-TOS-Versionen die Lage der Icons verschieben, kann man hier relativ sicher davon ausgehen, daß sie korrekt eingesetzt werden.

Es sei denn, die Images haben in neuen TOS-Versionen ein anderes Bitmuster oder sind nur noch im ROM zu finden.

Im ersten Fall brauchen nur die Data-Zeilen in den Satz-Prozeduren, die ja die Bitmuster enthalten, geändert zu werden. Im zweiten, unwahrscheinlichen Fall ließe sich dann allerdings nur noch mit EPROMs etwas ausrichten.

Mit der Prozedur 'P2' wird in Zusammenarbeit mit der Prozedur 'D_read' der komplette Deskicon- und maskensatz ausgewechselt.

Hier wird ebenfalls die eben beschriebene Suche durchgeführt.

```
Procedure P2
  A$=""
  Restore Desk_1_icn                  ! Data-Zeiger setzen
  For I%=0 To 31                      ! 32 Deskicon-Zeilen
    Read Dat%                         !  lesen
    A$=A$+Mkl$(Dat%)                  !  String bilden
  Next I%
  B$=Left$(A$,Len(A$)-1)              ! A$ auseinandernehmen, damit
  C$=Right$(A$)                       ! sich der String nicht im
  '                                   ! Speicher selber findet.
  A$=Space$(Len(A$))                  ! A$ wieder löschen
  @Find(B$,C$,2048,250000,2,*Adr%)    ! String im Speicher suchen
  If Adr%                             ! String gefunden?
    Print At(25,3);"Deskordner öffnen und OK klicken"
    Fileselect "\*.BI2","DESKTOPS",Desk$      ! Ordner öffnen
    Cls
    For I%=0 To 4                             ! 5 mal
      @D_read(Desk$+"DESKMSK"+Str$(I%+1)+".DZ2",Adr%+I%*256)
      @D_read(Desk$+"DESKICN"+Str$(I%+1)+".DZ2",Adr%+128+I%*256)
      ' Desktop-Masken und -Icons einlesen und eintragen
    Next I%
  Endif
  Desk_1_icn:
  Data 0,0,4064,6192,8351772,12681220,58785783,33554453,268172243,
134614103,1073481549,536871257,-463,-2147483037,-2147482939,-2147482743
  Data -2147482861,-2147483099,-2147483063,-2147482991,-2114059485,
-2130574778,-2130574708,-2114059496,-2147483088,-2147483040,
-2096758080,-2013527168,-2147482880,-2147483136,-2147483136,-512
Return
```

Mit dieser Prozedur wird in Zusammenarbeit mit der Prozedur 'D_read' der komplette Alert-Icon-Satz ausgewechselt.

```
Procedure P3
  A$=""
  Restore A.lert_1_icn                ! Data-Zeiger setzen
  For I%=0 To 31                      ! 16 Maus-Icon-Zeilen
    Read Dat%                         !  lesen
    A$=A$+Mkl$(Dat%)                  !  String bilden
  Next I%
  B$=Left$(A$,Len(A$)-1)              ! A$ auseinandernehmen, damit
  C$=Right$(A$)                       ! sich der String nicht im
  '                                   ! Speicher selber findet.
  A$=Space$(Len(A$))                  ! A$ wieder löschen
  @Find(B$,C$,2048,250000,3,*Adr%)    ! String im Speicher suchen
  If Adr%                             ! String gefunden?
    Print At(25,3);"Alertordner öffnen und OK klicken"
    Fileselect "\*.BI2","F_ALERTS",Alrt$    ! Ordner öffnen
    Cls
    For I%=0 To 2                           ! drei Alert-Icons
      @D_read(Alrt$+"ALRTICN"+Str$(I%+1)+".DZ2",Adr%+I%*128)! lesen/
      '                                                     ! setzen
    Next I%
  Endif
  A.lert_1_icn:
  Data 245760,417792,897024,1824768,3664896,7337472,14433024,29113728,
58474176,117194592,234635184,469516248,939278316,1878802422,-537116677,
-1073987587
  Data -1073987587,-537116677,1878802422,939278316,469762008,234880944,
117194592,58474176,29113728,14433024,7337472,3664896,1824768,897024,
417792,245760
Return
```

Die folgende Prozedur übernimmt das Lesen der 32-Bit-Daten aus den Dateien und das Übertragen der Daten in den AES-Icon-Speicher.

```
Procedure D_read(File$,Adress%)
  Open "I",#1,File$                   ! Datei öffnen
  Seek #1,2                           ! Filepointer auf 3. Byte
  '                                   ! setzen
  For J%=0 To 31                      ! 32 Zeilen
```

```
    If J%=16                          !  Data-Zeilenende?
      Relseek #1,2                    !   dann 2 Byte ('D ' als Abk.
      '                               !   für 'Data' am Zeilen-
      '                               !   anfang) überspringen.
    Endif
    Input #1;A%                       !   Icon/Masken-Zeile lesen
    Lpoke Adress%+J%*4,A%             !   in Speicher schreiben
  Next J%                             !  nächste Icon/Masken-Zeile
  Close #1
Return
Procedure P4
  @P1
  @P2
Return
Procedure P5
  @P1
  @P3
Return
Procedure P6
  @P2
  @P3
Return
Procedure P7
  @P1
  @P2
  @P3
Return
```

Wie oben bereits angedeutet, wird hier das jeweilige Bitmuster im Speicher gesucht und die gefundene Adresse an die Satz-Routine zurückgeliefert, die dann dort die Icons und Masken einsetzt.

Diese Prozedur eignet sich hervorragend als Utility zur Stringsuche im Speicher. In diesem Fall muß jedoch in 'Typ%' eine 3 übergeben werden, damit von der gefundenen Adresse kein Offset abgezogen wird.

Warum zwei Strings übergeben werden, wurde bereits beim Prozedur-Aufruf weiter oben erläutert. Mit den Parametern 'St%' und 'En%' können Sie auch den Bereich bestimmen, in

dem gesucht werden soll. In 'A_%' wird im Erfolgsfall die betreffende Adresse zurückgeliefert. Wurde der String nicht gefunden, wird in A_% eine Null zurückgegeben.

Diese Prozedur ist darauf ausgerichtet, daß die Suche bei dem ersten gefundenen String abgebrochen wird. Setzt man in der Abfrage 'If Ofs%' einen rekursiven Aufruf ein, z.B. '@Find(Str1$,Str2$,I%+Ofs%,En%,3,*A_%)', und davor evtl. ein 'Print I%+Ofs%-1' können so alle Strings mit dem gesuchten Inhalt im Speicher der Reihe nach ausfindig gemacht werden.

```
Procedure Find(Str1$,Str2$,St%,En%,Typ%,A_%)
  B$=Space$(32000)                  ! Puffer vorbereiten
  For I%=St% To En% Step 30000      ! Lese-Schleife
    Bmove I%,Varptr(B$),32000       !  Speicherbereich in Puffer
    Ofs%=Instr(B$,Str1$+Str2$)      !  Bitmuster darin enthalten?
    Exit If Ofs%                    !  Abbruch, wenn ja
  Next I%
  If Ofs%                           !  String gefunden?
    If Typ%=1                       ! Typ 1 (Maus-Icon)?
      *A_%=I%+Ofs%-180-1            !  Da der Eindeutigkeit halber
      '                             !  das 'Bienchen' gesucht wurde,
      '                             !  muß die Adresse auf den
      '                             !  Anfang des Pfeil-Maske zurück-
      '                             !  gesetzt werden (-180).
    Endif
    If Typ%=2                       ! Typ 2 (Deskicon)?
      *A_%=I%+Ofs%-128-1            !  Hier wurde nach dem Icon
      '                             !  des Disketten-Symbols gesucht.
      '                             !  Der Anfang der ersten Maske
      '                             !  liegt 128 Byte zurück (-128).
    Endif
    If Typ%=3                       ! Typ 3 (Alerticon)?
      *A_%=I%+Ofs%-1                !  Die Alert-Icons verfügen über
      '                             !  keine Maske, da sie nur auf
      '                             !  weißem Hintergrund verwendet
      '                             !  werden. Also braucht die Adresse
      '                             !  nicht reduziert zu werden.
    Endif
  Else                              ! String nicht gefunden
    *A_%=0                          !  Null zurückgeben
  Endif
Return
```

1.3 Images zum Selbermachen - ein Image-Editor

Ich habe nun keine Mühe gescheut, Ihnen die Möglichkeit zu eröffnen, sich eine eigene Programm-Umgebung zu schaffen.

Schauen Sie sich die folgenden vier Programme in Ruhe an. Mit diesen Programmen und Teilen daraus können Sie nun innerhalb Ihrer Programme gezielt verschiedene Alert-Symbole und beliebige Maus-Zeiger aufrufen. Das Besondere daran ist, wie vorher schon erwähnt, daß diese Icons fest in das GEM installiert sind, solange Sie nicht ausgewechselt werden bzw. nicht der RESET-Knopf gedrückt wird.

Sie können zwar auch über den 'DEFMOUSE X$'-Befehl eigene Mauszeiger entwerfen. Diese haben jedoch den Nachteil, daß Sie nur innerhalb laufender GFA-Programme zu verwenden sind. Gefällt Ihnen der schwarze Mauszeiger bei der Edition Ihrer Programme nicht, installieren Sie Ihren eigenen Maussatz. In Verbindung mit den selbst entworfenen Desktop-Icons können Sie dann auch nach Programmende Ihren persönlichen Eindruck hinterlassen.

Als erstes werde ich ein kleines Editor-Programm vorstellen, das speziell auf den Zweck des GEM-Icon-Austauschs zugeschnitten ist. Außerdem hoffe ich, Ihnen einen Einblick in den grundlegenden Aufbau eines modularen Editors sowie einige allgemein brauchbare Prozeduren anbieten zu können. Editoren haben eine wichtige Funktion in allen Programmen, die dem Anwender eigene Entwurfsmöglichkeiten zur Verfügung stellen sollen.

Ich habe diesen Editor gezielt so kurz wie möglich gehalten, damit er nicht zu kompliziert wird und Sie die Chance haben, ihn nach Ihren Bedürfnissen anpassen zu können.

Eine Schlüsselrolle bei der Anpassung spielen hier die Variablen:

M%	Matrixgröße
O%	Rand-Offset links u. oben
R%	Editorpunktgröße in Pixel

Mit Änderung dieser drei Variablen können Sie die Position, Anzahl der vertikalen/horizontalen Editorpunkte und die Größe der Punkte bestimmen.

```
' Programm: IMAGEEDIT.BAS
'
' |''''''
```

```
    R%=Int((400-O%*2)/M%)          ! R% = Editorpunktgröße in Pixel
    If B2%<>2                      ! Screen löschen ?
      @Drawgrid(M%,O%,R%)          !   ja
    Endif
    If B3%=1                       ! Image laden ?
      @Imageload(M%/B1%,M%,O%,R%) !   ja
    Endif
    @Setboxes(M%,O%,R%,0,*D1%,*D2%)! Image editieren (D1%/D2% =Dummies)
    Alert 2,"IMAGE speichern ?",2,"OKAY|NEIN",B4%
    If B4%=1                       ! Image speichern ?
      @Imagesave(M%/B1%,M%,O%,R%) !   ja
      If B6%=2                     !   Wurde Maus-Icon editiert ?
        @Get_hotspot(M%,O%,R%)     !     Aktionspunkt ?
      Endif
    Endif
    Alert 2,"Programmende ?",2,"OKAY|NEIN",B5%
    Exit If B5%=1
  Loop
Return
```

Nach Maus-Icon-Edition wird in der nächsten Prozedur der Aktionspunkt erfragt. Die lokalen Variablen 'Gr%', 'Os%', 'Fa%' haben in allen Prozeduren dieselbe Bedeutung:

Gr% Format (16 oder 32)
Os% Entfernung des Editorfeldes zum oberen und linken Rand
Fa% Editorpunktgröße in Pixel

```
Procedure Get_hotspot(Gr%,Os%,Fa%)
  For I%=0 To Gr%-1                  ! Alle Editor-Zeilen
    For J%=0 To Gr%-1                !   Alle Zeilen-Punkte
      If Point(Os%+J%*Fa%+Fa%/2,Os%+I%*Fa%+Fa%/2) ! Feld schwarz ?
        Deffill ,0,0                 !     dann markieren
        Pbox Os%+3+J%*Fa%,Os%+3+I%*Fa%,Os%-3+J%*Fa%+Fa%,Os%-3+I%*Fa%+Fa%
      Endif
    Next J%
  Next I%
  Alert 1,"Bitte Aktions-Punkt setzen !",1,"OKAY",Bb%
  @Setboxes(Gr%,Os%,Fa%,1,*Xa%,*Ya%) ! Aktionspunkt klicken
  Open "U",#99,Num_fl$               ! Maus-Icon-Datei öffnen
  Seek #99,Lof(#99)-2                ! Pointer auf File-Ende -(CR+LF)
  Print #99;",";Xa%;",";Ya%          !   Aktion-Koordinaten anhängen
  Close #99
```

```
Return
Procedure Drawgrid(Gr%,Os%,Fa%)       ! Zeichnen des Editorfeldes
  Local I%
  Color 1
  Graphmode 1
  Deffill ,2,4
  Pbox 0,0,639,399
  Print At(46,12);" Linke  Maustaste = Punkt  setzen "
  Print At(46,13);" Rechte Maustaste = Punkt löschen "
  Print At(46,14);" sonstige   Taste =       Abbruch "
  Deffill ,0,0
  Pbox Os%,Os%,Os%+Gr%*Fa%,Os%+Gr%*Fa%      !weißen Hintergrund
  Graphmode 1
  Pbox Os%+Gr%*Fa%+1,Os%,Os%+Gr%*Fa%+2+Gr%,Os%+Gr%+1
  Graphmode 3
  Pbox Os%+Gr%*Fa%+1,Os%,Os%+Gr%*Fa%+2+Gr%,Os%+Gr%+1
  Graphmode 1
  For I%=0 To Gr%                         ! Raster zeichnen
    Line Os%+I%*Fa%,Os%,Os%+I%*Fa%,Os%+Fa%*Gr%
    Line Os%,Os%+I%*Fa%,Os%+Fa%*Gr%,Os%+I%*Fa%
  Next I%
Return
```

Das Ermitteln des Editorpunkt-Indices und Zeichnen/Löschen der Editorpunkte wird in dieser Prozedur erledigt.

Fl% Flag für Aktionspunkt-Bestimmung bei Maus-Icons
Xb% Rückgabevariable an Get_hotspot
Yb% Rückgabevariable an Get_hotspot

```
Procedure Setboxes(Gr%,Os%,Fa%,Fl%,Xb%,Yb%)
  Local X%,Y%
  Graphmode 1
  Repeat                          ! Klick-Schleife
    Mouse X%,Y%,K%
    X%=Int((X%-Os%)/Fa%)          ! Editor-Index horizontal
    Y%=Int((Y%-Os%)/Fa%)          ! Editor-Index vertikal
    If X%=>0 And X%<Gr% And Y%=>0 And Y%<Gr% ! Im Editor ?
      If K%=1                     ! linke Maustaste ?
        Color 1
        Deffill ,1,1              ! schwarze Box setzen
        Pbox Os%+X%*Fa%,Os%+Y%*Fa%,Os%+X%*Fa%+Fa%,Os%+Y%*Fa%+Fa%
        Plot Os%+2+Gr%*Fa%+X%,Os%+1+Y%
```

```
      Endif
      If K%=2                          ! rechte Maustaste ?
        Color 0
        Deffill ,0,0                   ! weiße Box setzen
        Pbox Os%+X%*Fa%,Os%+Y%*Fa%,Os%+X%*Fa%+Fa%,Os%+Y%*Fa%+Fa%
        Plot Os%+2+Gr%*Fa%+X%,Os%+1+Y%
      Endif
    Endif
  Until Len(Inkey$) Or (Fl%=1 And K%) ! Abbruch, wenn Tastatur
  '                                   ! (oder Mausknopf nach
  '                                   ! Aktionspunkt-Abfrage)
  *Xb%=X%                             ! Rückgabe an Get_hotspot
  *Yb%=Y%                             !     "            "
Return
```

Diese Prozedur regelt das Einlesen der Musterdaten und die Übertragung in das Editor-Raster.

Te% Teiler zur Formatbestimmung im Dateinamen (16 oder 32)

```
Procedure Imageload(Te%,Gr%,Os%,Fa%)
  Local Fl$,El$,J%,I%
  @Head(1,"Image laden",0)
  Fileselect "\*.BI"+Str$(Gr%/Te%),".BI"+Str$(Gr%/Te%),Fl$
  @Head(0,"",0)
  If Exist(Fl$)                       ! Datei vorhanden ?
    Open "I",#99,Fl$                  ! Öffnen
    For I%=0 To Gr%-1                 ! Alle Editor-Zeilen
      Input #99,El$                   ! einlesen
      El$=Right$(El$,Gr%)             ! D für 'Data' abschneiden
      For J%=0 To Gr%-1               ! Alle Zeilenpunkte
        If Mid$(El$,J%+1,1)="1"       !   Bit gesetzt ?
          Color 1                     !   dann in Editor einsetzen
          Deffill ,1,1
          Pbox Os%+J%*Fa%,Os%+I%*Fa%,Os%+J%*Fa%+Fa%,Os%+I%*Fa%+Fa%
          Plot Os%+2+Gr%*Fa%+J%,Os%+I%+1
        Endif
      Next J%
    Next I%
    Close #99
  Endif
Return
```

Diese Prozedur erledigt die Muster-Speicherung auf Disk:

```
Procedure Imagesave(Te%,Gr%,Os%,Fa%)
  Local Fl$,El$,J%,I%
  @Head(1,"Image speichern",0)
  Fileselect "\*.BI"+Str$(Gr%/Te%),Fn$+"?.BI"+Str$(Gr%/Te%),Fl$
  @Head(0,"",0)
  If Fl$="" Or Fl$="\.BI"+Str$(Gr%/Te%) ! Ungültiger Dateiname ?
    Goto No.load                        !   dann zurück
  Endif
  If Instr(Right$(Fl$,4),".")=0         ! Extension gesetzt ?
    Fl$=Fl$+".BI"+Str$(Gr%/Te%)         !  Nein? Dann ergänzen
  Endif
  If Right$(Fl$,3)<>"BI"+Str$(Gr%/Te%)  ! Extension für Binärfile
    '                                   ! korrekt?
    Mid$(Fl$,Len(Fl$)-3,3)="BI"+Str$(Gr%/Te%) ! Nein? Korrigieren!
  Endif
  Num_fl$=Left$(Fl$,Len(Fl$)-3)+"DZ"+Str$(Gr%/Te%)
  '                                     ! Name für Dezimalfile
  Open "O",#98,Num_fl$                  ! Dezimal-File öffnen
  Open "O",#99,Fl$                      ! Binär-File öffnen
  For I%=0 To Gr%-1                     ! Alle Editor-Zeilen
    Clr El$
    For J%=0 To Gr%-1                   ! Alle Zeilen-Punkte
      If Point(Os%+J%*Fa%+Fa%/2,Os%+I%*Fa%+Fa%/2) ! Punkt gesetzt?
        El$=El$+"1"                     ! Ja? '1' in String setzen
      Else
        El$=El$+"0"                     ! Nein? '0' setzen
      Endif
    Next J%
    If I% Mod 16=0                      ! Zeilenanfang?
      Print #98;"D ";                   ! D für 'Data' schreiben
    Endif
    Print #98;Val("&X"+El$);            ! Dezimalwert schreiben
    If (I%+1) Mod 16=0                  ! Dezimal-Data-Zeilenende?
      Print #98                         ! Ja? CR / LF setzen
    Else
      Print #98;",";                    ! Nein? Komma setzen
    Endif
    Print #99;"D ";El$                  ! Binär-Data-Zeile schreiben
  Next I%
  Close #98
  Close #99
  No.load:
Return
```

Ein Utility zum Fileselect-Box-Aufruf

Hiermit wird ein Rahmen mit Text über die Box gezeichnet, was natürlich einen professionellen Eindruck beim Anwender hinterläßt und außerdem, was wohl wichtiger sein sollte, informativ ist.

Durch die Abfrage der Bildschirmauflösung und die Anpassung der Koordinaten kann dieses Utility auch in MIDRES verwendet werden.

Flg%	1=Box setzen / 0=Box löschen
Tx$	Boxtext
O%	Offset zum oberen Fensterrand bei 'Openw 0'

```
Procedure Head(Flg%,Tx$,O%)
  Local R%,H%,D%
  If Flg%=1                    ! 1 = Textbox zeichnen
    R%=3-Xbios(4)              ! R%=Teiler für Auflösung
    H%=Len(Tx$)*4              ! H%=halbe Textlänge
    D%=2-R%                    ! D%=Anpassung für HIRES
    Graphmode 1
    Deftext 1,1,,13
    Get 158,0,481,399/R%/6,Part.bk$ !Hintergrund abspeichern
    Deffill 1,2,2                  !graue Hintergrundbox
    Pbox 158,16/R%+D%*16-O%,481,(50/R%-D%*12)+D%*16-O%
    Deffill 1,0,0                  !weiße Textbox
    Pbox 320-H%-2,16/R%+D%*16-O%,320+H%+2,(50/R%-D%*12)+D%*16-O%
    Text 320-H%,(44/R%-D%*10)+D%*16-O%,Tx$
  Else                         ! wenn Flg%<>1
    Put 158,0,Part.bk$         ! Hintergrund wieder restaurieren
    Partbk$=""                 ! String löschen
  Endif
Return
```

1.4 Öfter mal was Neues

Mit dem Programm 'IMAGE-LOADER' haben Sie gelernt, wie man die einzelnen AES-Icon-Sätze 'en Block' austauscht. Damit haben Sie noch nicht die Möglichkeit, innerhalb Ihrer Programme ständig verschiedene Icons einzusetzen. Die folgenden beiden Programme erlauben es, beliebige Maus- oder Alert-Icons bei Bedarf einzusetzen.

Während Sie im Normalfall nur drei Alert-Icons zur Verfügung haben ('STOP', '!' und '?'), können Sie nun die Icons je nach Bedarf so oft wie nötig austauschen und immer wieder die Alertboxen gezielt grafisch ausstatten.

In Verbindung mit der AES-Form-Alertbox (s. unter 'AES') haben Sie somit ein breiteres Verwendungsspektrum für die Alertboxen.

1.4.1 Alert-Icons austauschen

```
' Programm: SET_ALERT.BAS
'
' |''''''
```

```
  For J%=1 To A%                    ! Alle neuen Alert-Icons
    @Setalert(J%,1)                 !   neues Icon setzen
    Alert 1,"Dies ist ein neues Icon !",1,"Oh,schön",Back%
  Next J%
Endif
```

In der obigen For/Next-Schleife werden die gelesenen Alert-Icons, die sich nun in dem zweidimensionalen Integer-Array 'A_plane%()' befinden, in den AES-Speicher eingetragen und in dem Alertbox-Aufruf angezeigt. Wurden die Alertdatas also in das Feld eingelesen, übernimmt die folgende Prozedur das Installieren der Alert-Icons.

Die ausgetauschten Icons bleiben auch nach Programmende im Speicher und können erst wieder durch erneuten Austausch oder RESET gelöscht werden.

Index% gibt an, welches neue Alert-Image aus dem Feld verwendet werden soll (Startindex = 1).

Nummer% ist der Icon-Index der Alertbox-Variante, in welcher Sie das neue Icon verwenden wollen.

Alert 1,..... = Nummer 1
Alert 2,..... = Nummer 2
Alert 3,..... = Nummer 3

```
Procedure Setalert(Index%,Nummer%)
  For I%=0 To 31                        ! 32 Icon-Zeilen
    Lpoke Adr%+(Nummer%-1)*128+I%*4,A_plane%(Index%,I%) ! in Speicher
    '                                                   ! lpoken
  Next I%
Return
```

In der nächsten Prozedur werden die Datas des Alert-Icons eingelesen und in das zweidimensionale Array übertragen. Die erste Ebene dieses Feldes enthält den Feldindex des Alert-Icons, während die zweite die Icon-Daten enthält.

Anzahl% beliebiger Wert, der die maximal zu lesenden Alert-Icons angibt. Wird eine Null übergeben, werden solange Datas gelesen, bis die Endmarkierung '**' erreicht wird.

Ret% Rückgabewert. Enthält die tatsächlich geladene Anzahl.

```
Procedure Alert_read(Anzahl%,Ret%)
  If Anzahl%=0                    ! Anzahlübergabe = 0?
    Anzahl%=20                    !  max. Anzahl bestimmen (beliebig)
  Endif
  Local A_nummer%,J%
  Erase A_plane%()                !  Feld löschen
  Dim A_plane%(Anzahl%,31)        !  Feld dimensionieren
  Restore A_datas                 !  Data-Zeiger setzen
  Do                              !  Lese-Schleife
    Inc A_nummer%                 !   Index-Zähler +1
    For J%=0 To 31                !   32 Datas
      Read Bits$                  !    Data lesen
      Exit If Bits$="**" Or (Anzahl%>0 And A_nummer%=Anzahl%)
      '                           !    Abbruch wenn Endmarkierung oder
      '                           !    max. vorgegebene Anzahl erreicht
      A_plane%(A_nummer%,J%)=Val(Bits$) !Data in Feld eintragen
```

Wollen Sie statt der Dezimaldatas lieber die Binärdatas verwenden, brauchen Sie in der vorstehenden Programmzeile nur den Ausdruck hinter dem Gleichheitszeichen gegen 'Val("&X"+Bits$)' auszutauschen.

```
    Next J%                       !  nächstes Data
    Exit If Bits$="**" Or (Anzahl%>0 And A_nummer%=Anzahl%)
    '                             !  Abbruch wenn Endmarkierung oder
    '                             !  max. vorgegebene Anzahl erreicht
  Loop
  *Ret%=A_nummer%-1               ! tatsächlich gelesene Anzahl
    '                             ! zurückgeben
Return
Procedure Find(Str1$,Str2$,St%,En%,A_%)
  B$=Space$(32000)                ! Puffer vorbereiten
```

```
  For I%=St% To En% Step 30000        ! Lese-Schleife
    Bmove I%,Varptr(B$),32000         !  Speicherbereich in Puffer
    Ofs%=Instr(B$,Str1$+Str2$)        !  Bitmuster darin enthalten?
    Exit If Ofs%                      !  Abbruch, wenn ja
  Next I%
  If Ofs%
    *A_%=I%+Ofs%-1                    !  Alert-Icon-Adresse zurückgeben
  Else
    *A_%=0
  Endif
Return
A_datas:
Data 268431360,469769216,806094592,1611792768,1276248256,-1845355424,
-2113527264,-2113142224,-1944583664,-1877992432,-1844305904,
-1944968688,-2146690032,-872415216,-872409040,1611405408
Data 1879838656,939528288,1040195616,532684832,134209568,16773152,14576,
8088,3976,-1879023160,-1751248648,-178970512,-1789575168,-1789579264,
-1789591552,8192
Data 1984,1771826272,1800985136,2077807632,1800984592,1801376784,6160,
250896,489488,791584,3938336,15361056,25964576,21633088,26040384,
21474300
Data 128162822,231858178,216955907,175687169,203407889,167921321,
221936977,111307441,62799841,66846721,60841985,22041602,27978414,
13989212,7076600,2033600
Data 16777152,64312672,113245872,91226968,111393192,98438872,109052008,
94322264,523314046,828848611,1305162991,2077105401,1774980301,
1221332165,1220807109,1219758917
Data 1218186821,1820592205,646187851,814619078,478884044,126476920,
8929856,8930880,11013184,11303232,6783360,3160832,1369600,919040,461824,
129024
Data **
A.lert_1_icn:
Data 245760,417792,897024,1824768,3664896,7337472,14433024,29113728,
58474176,117194592,234635184,469516248,939278316,1878802422,-537116677,
-1073987587
Data -1073987587,-537116677,1878802422,939278316,469762008,234880944,
117194592,58474176,29113728,14433024,7337472,3664896,1824768,897024,
417792,245760
```

1.4.2 Maus-Icons austauschen

Mit diesem Programm haben Sie die Möglichkeit, sich eigene Mauszeigersätze im AES anzeigen zu lassen. Dazu müssen die Maus-Icons allerdings mit 'IMAGEDIT.BAS' bearbeitet bzw. zumindest abgespeichert worden sein. Außerdem ist zu jedem Icon die entsprechende Mausmaske zu erstellen. In den Data-Zeilen am Programmende müssen im Wechsel immer zuerst die Mausmaskendaten und dann die jeweiligen Icon-Daten stehen, wobei die Maus-Icon-Data-Zeilen je 18 Werte enthalten. Als Endmarkierung für den gesamten Data-Block ist das '*'-Zeichen zu verwenden (s.unten).

Um den Mauszeiger direkt zu ändern, benutzen Sie folgenden Trick: Starten Sie das Programm und klicken Sie dann so oft mit der Maus, bis die gewünschte Maus erscheint. ANschließend brechen Sie das Programm mit <Ctrl>+<Shift>+<Alt> ab.

```
' Programm: SET_MAUS.BAS
'
' |""""""""""""""""""""""""""""""""""""""""""|
' |              MAUS - SETTING               |
' |-------------------------------------------|
' |___________________________________________|

A$=""
Restore Maus_0_icn                    ! Data-Zeiger setzen
For I%=0 To 15                        ! 16 Maus-Icon-Zeilen
  Read Dat%                           !  lesen
  A$=A$+Mki$(Dat%)                    !  String bilden
Next I%
B$=Left$(A$,Len(A$)-1)                ! A$ auseinandernehmen, damit
C$=Right$(A$)                         ! sich der String nicht im
'                                     ! Speicher selber findet.
A$=Space$(Len(A$))                    ! A$ wieder löschen
@Find(B$,C$,2048,250000,*Adr%)        ! String im Speicher suchen
If Adr%                               ! String gefunden?
  @Maus_read(0,*A%)                   ! Mausdatas lesen
  Print "Maustaste drücken"
  For J%=1 To A%                      ! Alle neuen Mäuse
    Repeat
    Until Mousek                      ! Auf Mausklick warten
    @Setmaus(J%,0)                    !  neue Maus setzen
```

```
    Defmouse 0
    Pause 20
  Next J%
Endif
```

Wurden die Mausdatas in das Feld eingelesen, können Sie mit dieser Prozedur die Mauszeiger austauschen. Der Vorteil gegenüber der 'DEFMOUSE Maus$'-Funktion ist hierbei, daß diese Mauszeiger auch innerhalb von Fileselect- und Alertboxen bzw. auch im Interpreter verwendet werden können. Dazu muß das gewünschte Icon mit 'DEFMOUSE 0' aufgerufen werden.

Bei den Mäusen ist es wichtig zu wissen, daß die Angaben über den Aktionspunkt, also der Punkt im Mausbild, auf den sich die Aktionen (Mausklick / Punkt fixieren) beziehen sollen, genau 10 Bytes vor der dazugehörigen Mausmaske im Speicher beginnen. Und zwar die X-Koordinate innerhalb des Mausbildes (0-15) in Mausmaskenadresse minus 10 und die Y-Koordinate (0-15) in Mausmaskenadresse minus 8.

Index% gibt an, welches neue Maus-Image aus dem Feld verwendet werden soll.

Nummer% ist der Defmouse-Index, mit welchem das Maus-Icon später aufgerufen werden kann.

```
Procedure Setmaus(Index%,Nummer%)
  For I%=0 To 31                          ! 32 Zeilen (Maske und Icon)
    Dpoke Adr%+Nummer%*74+I%*2,M_plane%(Index%,I%) ! in Speicher dpoken
  Next I%
  Lpoke Adr%-10+Nummer%*74,M_plane%(Index%,32)*65536+M_plane%(Index%,33)
  '                                       ! Aktionspunktkoord. in Speicher
Return
```

In dieser Prozedur werden die Mausdatas (zuerst Mausmaske, dann Icon und Aktionpunktkoordinaten) eingelesen und in ein zweidimensionales Array übertragen. Die erste Ebene dieses Feldes enthält die Nummer des Mauszeigers, während die zweite die Mausdaten enthält.

Anzahl% beliebiger Wert, der die maximal zu lesenden Maussätze angibt. Wird eine Null übergeben, werden solange Datas gelesen, bis die Endmarkierung '*' erreicht wird.

Ret% Rückgabewert. Enthält die tatsächlich geladene Anzahl.

```
Procedure Maus_read(Anzahl%,Ret%)
  If Anzahl%=0                    ! Anzahlübergabe = 0?
    Anzahl%=20                    !  max. Anzahl bestimmen (beliebig)
  Endif
  Local M_nummer%,J%
  Erase M_plane%()                !  Feld löschen
  Dim M_plane%(Anzahl%,33)        !  Feld dimensionieren
  Restore M_datas                 !  Data-Zeiger setzen
  Do                              !  Lese-Schleife
    Inc M_nummer%                 !   Index-Zähler +1
    For J%=0 To 33                !   34 Datas
      '                           !   (16 für Maske + 16 für Icon + 2
      '                           !   für Aktionspunkt)
      Read Bits$                  !    Data lesen
      Exit If Bits$="*" Or (Anzahl%>0 And M_nummer%=Anzahl%)
      '                           !    Abbruch wenn Endmarkierung oder
      '                           !    max. vorgegebene Anzahl erreicht
      M_plane%(M_nummer%,J%)=Val(Bits$) !Data in Feld eintragen
    Next J%                       !  nächstes Data
    Exit If Bits$="*" Or (Anzahl%>0 And M_nummer%=Anzahl%)
    '                             !  Abbruch wenn Endmarkierung oder
    '                             !  max. vorgegebene Anzahl erreicht
  Loop
  *Ret%=M_nummer%-1               ! tatsächlich gelesene Anzahl
    '                             ! zurückgeben
Return
```

Die Prozedur 'Find' ist Ihnen oben schon einmal begegnet.

```
Procedure Find(Str1$,Str2$,St%,En%,A_%)
  B$=Space$(32000)                ! Puffer vorbereiten
  For I%=St% To En% Step 30000    ! Lese-Schleife
    Bmove I%,Varptr(B$),32000     !  Speicherbereich in Puffer
    Ofs%=Instr(B$,Str1$+Str2$)    !  Bitmuster darin enthalten?
    Exit If Ofs%                  !  Abbruch, wenn ja
```

```
  Next I%
  If Ofs%
    *A_%=I%+Ofs%-180-1              !  1. Mausmasken-Adresse zurückgeben
  Else
    *A_%=0
  Endif
Return
M_datas:
' Mausindex 1 (Maske)
Data 65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,
65535,65535,65535,65535,65535
' Mausindex 1 (Icon)
Data 65535,32769,40953,53235,42981,54219,43413,54315,43605,54891,44085,
55323,45069,57351,49155,65535,7,8
' Mausindex 2 (Maske)
Data 2016,6168,8772,20082,23130,37449,48765,32769,32769,48765,37449,
23130,20082,8772,6168,2016
' Mausindex 2 (Icon)
Data 0,2016,7608,12684,8580,24966,16770,32382,32382,16770,24966,8580,
12684,7608,2016,0,7,7
' Mausindex 3 (Maske)
Data 65472,65472,65472,65472,65534,65534,65534,65534,65534,65535,4095,
4095,4095,4095,4095,127
' Mausindex 3 (Icon)
Data 59392,44928,59520,3712,65152,24316,24196,16628,32660,1364,1303,
1524,1031,2045,41,47,1,1
' Mausindex 4 (Maske)
Data 65520,65520,65520,65504,65472,65504,65520,65528,65532,65534,63487,
58367,511,255,127,62
' Mausindex 4 (Icon)
Data 0,32736,27328,21888,27392,21888,27328,23904,30384,25432,16812,214,
106,54,28,0,1,1
' u.s.w
Data *
Maus_0_icn:
Data 2048,2108,98,1730,50820,6538,6996,1760,7512,13308,24928,17118,
17624,19030,13332,0,
```

1.4.3 Aus Data mach' PUT

Mit dem 'MEMO_SPY.BAS' sowie dem 'IMAGEDIT.BAS' haben Sie die Möglichkeit, 16- bzw. 32-Bit-Images als Data-Zeilen auf Diskette abzulegen. Eine Verwendungsmöglichkeit dieser Bildchen fanden Sie in den vorangegangenen drei Programmen.

Dieses kleine Programm zeigt Ihnen eine weitere Verwendung dieser Images in eigenen Programmen. Es wäre doch schade, wenn Sie sich die Arbeit machen, eigene Icons zu entwerfen und diese 'nur' als Mauszeiger, Desktop- oder Alert-Icons einsetzen können. Mit dieser Routine werden genau dieselben Data-Zeilen, die oben verwendet werden, in Get/Put-Strings umgewandelt, die Sie dann durch den 'PUT'-Befehl beliebig auf dem Bildschirm plazieren oder weiterverarbeiten können.

Abb. 3: Icons über Icons

Bei den Data-Blöcken ist hier zu beachten, daß jeweils komplette Data-Sätze nach Formaten getrennt geladen werden. Die Data-Zeile 'Data *' gilt dabei als Endmarkierung für einen 16-Bit-Image-Block und die Zeile 'Data **' als Endmarkierung für

einen 32-Bit-Block. Der Blockstart kann beliebig durch Angabe von Sprung-Labels (hier: L_bin_datas / L_dez_datas / W_bin_datas / W_dez_datas) bestimmt werden. Die zusammengehörigen Image-Zeilen werden hier in einem zweidimensionalen Feld abgelegt, dessen erste Ebene die Image-Nummer und dessen zweite Ebene die dazugehörigen Image-Zeilen beinhaltet.

Die Erstellung des PUT-Strings findet in der Prozedur 'Make_icon' statt.

Bei Dezimal-Maus-Icons ('.DZ1'), die bereits mit dem 'IMAGEDIT.BAS' bearbeitet wurden (!) und hier als PUT-String benutzt werden sollen, müssen die letzten beiden Data-Werte der Dezimal-Icon-Data-Zeile gelöscht werden, da sonst der Data-Zähler falsche Werte liefert. Die Zeile darf also nur 16 Werte enthalten!

Übrigens haben Sie die Möglichkeit, sich Images aus fremden Programmen (sofern sie nicht ausdrücklich geschützt sind) zu 'leihen', indem Sie nach Beendigung des fremden Programms den 'MEMO_SPY.BAS' starten, sich die evtl. noch im Speicher befindlichen Images und Screens suchen und mit der 'Imagesave'-Funktion die interessanten Teile abspeichern. Dies funktioniert allerdings nicht immer. Der Interpreter überschreibt meist den Bereich des vorherigen Programms. In Ausnahmefällen lassen sich Screen-Teile direkt unterhalb des Bildschirmspeichers finden, wo sie bei Fileselect- und Alert-Box-Aufrufen abgelegt werden. Verfügen Sie über einen Compiler und compilieren den Speicher-Spion, ist die Chance, damit größere Screen-Teile alter Programme zu finden, erheblich größer.

```
' Programm: BIT_READ.BAS
'
' |''''''
```

```
@W_bitread(0,1,*Ptr%)          ! Word-Binär-Datas lesen
Lpoke Varptr(A%(0)),Ptr%       ! Anzahl in Feld eintragen
@L_bitread(0,1,*Ptr%)          ! Longword-Binär-Datas lesen
Lpoke Varptr(A%(1)),Ptr%       ! Anzahl in Feld eintragen
@W_bitread(0,2,*Ptr%)          ! Word-Dezimal-Datas lesen
Lpoke Varptr(A%(2)),Ptr%       ! Anzahl in Feld eintragen
@L_bitread(0,2,*Ptr%)          ! Longword-Dezimal-Datas lesen
Lpoke Varptr(A%(3)),Ptr%       ! Anzahl in Feld eintragen
'
Print At(1,1);A%(0);" Mausklicks, bitte !    "
For I%=1 To A%(0)              ! Alle Word-Binär-Icons
  @Make_icon(1,1,I%)           !  bauen und
  @Wait                        !  auf Mausklick warten und zeigen
Next I%
Print At(1,1);"  ";A%(1);" Mausklicks, bitte !    "
For I%=1 To A%(1)              ! Alle Longword-Binär-Icons
  @Make_icon(2,1,I%)           !  bauen und
  @Wait                        !  auf Mausklick warten und zeigen
Next I%
Print At(1,1);"    ";A%(2);" Mausklicks, bitte !    "
For I%=1 To A%(2)              ! Alle Word-Dezimal-Icons
  @Make_icon(1,2,I%)           !  bauen und
  @Wait                        !  auf Mausklick warten und zeigen
Next I%
Print At(1,1);"      ";A%(3);" Mausklicks, bitte !    "
For I%=1 To A%(3)              ! Alle Longword-Dezimal-Icons
  @Make_icon(2,2,I%)           !  bauen und
  @Wait                        !  auf Mausklick warten und zeigen
Next I%
Print At(1,1);"beliebige Taste = Ende                "
Void Inp(2)
Edit
```

Nach Aufbau des PUT-Strings wird hier auf einen Mausklick gewartet und das Image an der Mausposition gezeichnet. Diese Prozedur ist allerdings nur in diesem Beispielprogramm sinnvoll, da Sie sonst diese Images wahrscheinlich vom Programm aus plazieren werden.

```
Procedure Wait
  Repeat
  Until Mousek              ! auf Mausklick warten
```

```
  Put Mousex,Mousey,A$        ! Icon zeichnen
  Pause 20
Return
```

Im Zusammenhang mit der nächsten Prozedur ist es interessant zu erfahren, wie ein Get/Put-String aufgebaut ist.

Bei einem Monochrom-String ist es sehr einfach. Die ersten 6 Bytes des Strings enthalten drei Words, die der Reihe nach die Differenz der beiden gegenüberliegenden X-Koordinaten, die Differenz der beiden Y-Koordinaten und die Anzahl der Farbebenen des Ausschnitts enthalten. Im HIRES-Modus gibt es nur eine Farbebene. D.h., ein Pixel auf dem Bildschirm entspricht einem Bit im String. Ist ein Bildschirmpunkt schwarz, wird das entsprechende Bit im String gesetzt.

Beispiel:

Es soll ein Bildausschnitt mit X1/Y1 = 10/12 und X2/Y2 = 55/34 gespeichert werden. Die ersten drei Words des Strings ergeben sich aus Mki$(45) (= X2-X1) + Mki$(22) (= Y2-Y1) + Mki$(1) (= 1 HIRES-Farbebene). Nun wird die Pixel-Breite durch 16 geteilt. Das Ergebnis ist dann der benötigte Speicherplatz in Words pro Zeile. Bleibt ein Rest, wird ein Word hinzugerechnet. 46 / 16 = 2.875, also werden pro Zeile drei Words benötigt. Bei 23 Zeilen sind das dann insgesamt 23 * 3 = 69 Words + 3 Headerwords = 72 Words (144 Bytes). In diesem Fall würde also das erste Pixel der ersten Zeile durch das erste Bit des ersten Words repräsentiert, das erste Pixel der zweiten Zeile durch das erste Bit des vierten Words, das zweite Pixel der ersten Zeile durch das zweite Bit des ersten Words, das siebzehnte Pixel der ersten Zeile durch das 1 Bit im zweiten Word usw. Wer nun mitgerechnet hat, wird sich evtl. fragen, was mit den letzten 2 Bits jedes dritten Words passiert. Diese beiden Bits können getrost ignoriert werden, da sie keine verwertbare Information beinhalten.

Würde man einen senkrechten, ein Pixel breiten und 200 Pixel hohen Bildausschnitt ausschneiden, bräuchte man dafür immerhin 200 Words (400 Bytes) + 3 Headerwords = 406 Byte Spei-

cherplatz, obwohl nur das erste Bit jedes Words im String benutzt würde. Ein sechzehn Pixel breiter Streifen mit gleicher Höhe würde ebenfalls 'nur' 406 Bytes benötigen, wobei alle 16 Bits jedes Words belegt wären.

In MIDRES oder HIRES sieht das Ganze auf den ersten Blick etwas komplizierter aus. Wenn man das Prinzip jedoch erstmal begriffen hat, ist es fast genauso einfach.

Da im Farbmodus jedes Pixel unterschiedliche Farben annehmen kann, müssen die Informationen über die Farbe jedes einzelnen Pixel natürlich irgendwo gespeichert werden. Da nun aber ein einzelnes Bit nur zwei Zustände darstellen kann (1 oder 0), brauchen wir noch weitere Bits, um mehrere Farben darstellen zu können. Im MIDRES-Modus brauchen wir nur ein zusätzliches Bit und im LOWRES-Modus drei. Wie Sie wissen, kann man in MIDRES über vier verschiedene Farben gleichzeitig verfügen und in LOWRES über 16.

Um die Zahlen 0 - 3 darstellen zu können, benötigen wir 2 Bits (00 / 01 / 10 / 11 = 0,1,2,3). Mit diesen vier Zahlen wird nun das Farbregister bestimmt, aus welchem das Pixel seine Farbe bezieht. In MIDRES stehen immer nur die ersten 4 Farbregister (0 - 3) zur Verfügung. Im LOWRES-Modus geht das genauso vor sich, nur daß wir, um die Zahlen 0 bis 15 darstellen zu können, vier Bits benötigen (0000 / 0001 / 0010 / 0011 / 0100 / 0101 / 0110 / 0111 / 1000 / 1001 / 1010 / 1011 / 1100 / 1101 / 1110 / 1111 = 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15). Also wird hier mit einer der 16 Zahlen eines der 16 möglichen Farbregister bestimmt, aus dem das Pixel seine Farbe bezieht.

In diesem Mehrbedarf an Bits pro Pixel ist der Grund zu finden, warum im MIDRES-Modus nur die Hälfte und im LOWRES-Modus nur ein Viertel des Monochrom-Bildschirms zur Verfügung steht. Die Organisation des Bildschirmspeichers ist nun im Farbmodus so, daß für jeweils 16 waagerecht nebeneinander liegende Pixel des Bildschirms zwei (MIDRES), bzw. vier (LOWRES) Words hintereinander liegen. Die Farbe des ersten

Pixel ergibt sich aus der Bitkombination der ersten Bits der zugehörigen Words, die Farbe des zweiten aus der Kombination des zweiten Bits der zugehörigen Words usw.

Beispiel:

Pixel 0/0 soll im LOWRES-Modus die Farbe des 11. Farbregister annehmen. Die Zahl 9 wird mit vier Bits so dargestellt: 1011. D.h., im Video-RAM müssen dazu die obersten Bits der ersten vier Words folgendermaßen gesetzt sein:

Bit 1 von Xbios(2) ist gesetzt, Bit 1 von Xbios(2)+2 ist gesetzt, Bit 1 von Xbios(2)+4 ist nicht gesetzt und Bit 1 von Xbios(2)+6 ist wieder gesetzt.

Dazu ein kleines Beispielprogramm:

```
' Programm: BSP1_4_3.BAS

For I%=0 To 200
  Dpoke Xbios(2)+I%*160+I%*8,65535
  Dpoke Xbios(2)+2+I%*160+I%*8,65535
  Dpoke Xbios(2)+4+I%*160+I%*8,0
  Dpoke Xbios(2)+6+I%*160+I%*8,65535
Next I%
```

Sie sehen, daß das Bild mit einer Abstufung von 16 Pixel diagonal in der Farbe des Farbregisters 11 gestreift ist.

Genau nach demselben Schema ist auch der Get/Put-String im Farbmodus organisiert. Dabei enthält im MIDRES-Modus das dritte Word des Strings eine 2 (2 Ebenen) und im LOWRES-Modus eine 4 (4 Ebenen).

In der folgenden Prozedur werden nun Monochrom-Get/Put-Strings produziert. Die zu übergebenden Parameter haben folgende Bedeutung:

Format% 1 (16-Bit-Image)
2 (32-Bit-Image)

Flag% 1 (Binärdatas)
2 (Dezimaldatas)

Index% Image-Nummer

```
Procedure Make_icon(Format%,Flag%,Index%)
  A$=Mki$(Format%*16-1)+Mki$(Format%*16-1)+Mki$(1)   ! Breite, Höhe und
  '                                                  ! Auflösung
  '                                                  ! eintragen
  For J%=0 To Format%*16-1                           ! alle Icon-Zeilen
    If Format%=1                                     ! 16Bit-Image?
      If Flag%=1                                     !  Binär?
        A$=A$+Mki$(Val("&X"+W_plane.bin$(Index%,J%)))!   Zeile einbauen
      Else                                           !  Dezimal!
        A$=A$+Mki$(Val("&X"+W_plane.dez$(Index%,J%)))!   Zeile einbauen
      Endif
    Else                                             ! 32Bit-Image!
      If Flag%=1                                     !  Binär
        A$=A$+Mkl$(Val("&X"+L_plane.bin$(Index%,J%)))!   Zeile einbauen
      Else                                           !  Dezimal!
        A$=A$+Mkl$(Val("&X"+L_plane.dez$(Index%,J%)))!   Zeile einbauen
      Endif
    Endif
  Next J%                                            ! nächste Zeile
Return
```

In dieser Prozedur werden alle Word-Image-Datas (sofern vorhanden) eingelesen und in ein zweidimensionales Feld übertragen. Die erste Feld-Ebene enthält die Nummer des gelesenen Image, die zweite die dazugehörigen Zeilen.

Grundätzlich handelt es sich um dieselbe Prozedur, die bereits weiter oben ('Maus_read') beschrieben wurde, nur daß hier durch Angabe eines Flags bestimmt werden kann, ob Dezimal- oder Binärdatas gelesen werden sollen.

Anzahl% Anzahl der zu lesenden Word-Images

Flag% 1 (Binärdatas)
2 (Dezimaldatas)

Ret% Rückgabevariable für tatsächliche Image-Anzahl

```
Procedure W_bitread(Anzahl%,Flag%,Ret%)
  If Anzahl%=0                    ! Anzahlübergabe = 0?
    Anzahl%=20                    !  max. Anzahl bestimmen (beliebig)
  Endif
  Local W_nummer%,J%
  If Flag%=1                      ! Binärdatas?
    Erase W_plane.bin$()          !  Feld löschen
    Dim W_plane.bin$(Anzahl%,15)  !  Feld dimensionieren
    Restore W_bin_datas           !  Data-Zeiger setzen
  Else                            ! Dezimaldatas!
    Erase W_plane.dez$()          !  Feld löschen
    Dim W_plane.dez$(Anzahl%,15)  !  Feld dimensionieren
    Restore W_dez_datas           !  Data-Zeiger setzen
  Endif
  Do                              ! Lese-Schleife
    Inc W_nummer%                 !  Index-Zähler +1
    For J%=0 To 15                !  16 Zeilen
      Read Bits$                  !  Data lesen
      Exit If Bits$="*" Or (Anzahl%>0 And W_nummer%=Anzahl%)
      '                         !  Abbruch wenn Endmarkierung oder
      '                         !    max. vorgegebene Anzahl erreicht
      If Flag%=1                !  Binärdatas?
        W_plane.bin$(W_nummer%,J%)=Bits$ ! Zeile in Feld eintragen
      Else                      !  Dezimaldatas!
        W_plane.dez$(W_nummer%,J%)=Bin$(Val(Bits$)) ! Zeile in Feld
      '                                             ! eintragen
      Endif
    Next J%                     ! nächste Zeile
    Exit If Bits$="*" Or (Anzahl%>0 And W_nummer%=Anzahl%)
    '                           ! Abbruch wenn Endmarkierung oder
    '                           ! max. vorgegebene Anzahl erreicht
  Loop
  *Ret%=W_nummer%-1             ! tatsächlich gelesene Anzahl zurückgeben
Return
```

Auch diese Prozedur ist Ihnen in etwas abgewandelter Form oben schon begegnet. Hier werden alle Longword-Image-Datas (sofern vorhanden) eingelesen und in ein zweidimensionales Feld übertragen. Die erste Feld-Ebene enthält wieder die Nummer des gelesenen Images, die zweite die dazugehörigen Zeilen.

Anzahl% Anzahl der zu lesenden Word-Images

Flag% 1 (Binärdatas)
2 (Dezimaldatas)

Ret% Rückgabevariable für tatsächlich gelesene Image-Anzahl

```
Procedure L_bitread(Anzahl%,Flag%,Ret%)
  If Anzahl%=0                    ! Anzahlübergabe = 0?
    Anzahl%=20                    !  max. Anzahl bestimmen (beliebig)
  Endif
  Local L_nummer%,J%
  If Flag%=1                      ! Binärdatas?
    Erase L_plane.bin$()          !  Feld löschen
    Dim L_plane.bin$(Anzahl%,31)  !  Feld dimensionieren
    Restore L_bin_datas           !  Data-Zeiger setzen
  Else                            ! Dezimaldatas!
    Erase L_plane.dez$()          !  Feld löschen
    Dim L_plane.dez$(Anzahl%,31)  !  Feld dimensionieren
    Restore L_dez_datas           !  Data-Zeiger setzen
  Endif
  Do                              ! Lese-Schleife
    Inc L_nummer%                 !  Index-Zähler +1
    For J%=0 To 31                !  31 Zeilen
      Read Bits$                  !  Data lesen
      Exit If Bits$="**" Or (Anzahl%>0 And L_nummer%=Anzahl%)
      '                           !  Abbruch wenn Endmarkierung oder
      '                           !    max. vorgegebene Anzahl erreicht
      If Flag%=1                  !  Binärdatas?
        L_plane.bin$(L_nummer%,J%)=Bits$ ! Zeile in Feld eintragen
      Else                        !  Dezimaldatas!
        L_plane.dez$(L_nummer%,J%)=Bin$(Val(Bits$)) ! Zeile in Feld
        '                                           ! eintragen
      Endif
    Next J%                       ! nächste Zeile
    Exit If Bits$="**" Or (Anzahl%>0 And L_nummer%=Anzahl%)
    '                             ! Abbruch wenn Endmarkierung oder
    '                             !  max. vorgegebene Anzahl erreicht
  Loop
  *Ret%=L_nummer%-1               ! tatsächlich gelesene Anzahl
  '                               ! zurückgeben
Return
```

An dieser Stelle befinden sich im Programm (auf Disk) mehrere Maus- und Alert-Images als Binärdatas. Aus Gründen der Platzersparnis habe ich sie hier im Buch weggelassen.

1.5 Spezial-Adressen

Bei näherer Betrachtung der Speicher-Aktivitäten haben wir festgestellt, daß z.B. das AES in ganz bestimmten Bereichen seine Images ablegt und sie Bedarf von dort holt, um sie auf dem Bildschirm anzuzeigen.

Das ist aber bei weitem nicht alles, was man durch den Speicher-Spion erfahren kann. Es gibt so allerhand unscheinbare Adressen, die bei genauem Hinschauen sehr interessante Aktivitäten entwickeln.

Ich habe nun hier eine Sammlung von Adressen zusammengestellt, die Ihnen in vielen Beziehungen das Programmiererleben leichter machen können oder Effekte erlauben, die schon den Anschein von echter 'Intelligenz' Ihrer Programme aufkommen lassen.

Ich gehe bei all diesen Adressen immer davon aus, daß die normale Adressenlage nicht durch residente Programme, die im Auto-Ordner mitgebootet wurden, verschoben wurde. Wenn dieser seltene Fall auftreten sollte, kann ich für die richtige Funktion der aufgeführten Adressen leider nicht garantieren. Ob die Adressenlage verschoben wurde, läßt sich am leichtesten durch eine Positionsabfrage der Images feststellen, wie ich sie Ihnen weiter oben in der Prozedur 'Find' gezeigt habe. Stimmen die Image-Adressen, kann davon ausgegangen werden, daß alle anderen Adressen auch zutreffen. Der Ausnahmefall ist allerdings immer das Disketten-TOS. Durch den vom TOS in diesem Fall belegten Speicher werden die Adressen ganz erheblich (ca. 200 KByte) verschoben.

Dafür haben die (noch) Disketten-TOS-Besitzer die bestechende Möglichkeit, direkt auf den Zeichensatz-Speicher zugreifen zu

können. Der Entwurf eigener Schriftzeichen, die dann auch noch durch 'DEFTEXT' variiert werden können, ist dadurch in greifbare Nähe gerückt.

Warum man diesen nur 4096 Byte großen Bereich nicht genauso wie die AES-Images beim ROM-TOS in das RAM ausgelagert hat, ist mir nicht ganz verständlich. Man hätte dadurch noch wesentlich effektiver die Möglichkeit, seinen Programmen ein besonderes Aussehen zu verleihen. Leider haben die Leute von Digital Research dieses nicht für notwendig gehalten.

Der 8*16-Font ist folgendermaßen organisiert:

Es gibt 256 Schriftzeichen, von denen zwei einfach leer sind. Das sind die ASCII-Zeichen 0 (NULL) und 32 (SPACE). Jedes andere ist aus 16 Bitzeilen zusammengesetzt, wovon jede Zeile ein Byte umfaßt. Im Font-Speicher liegt die erste Bitzeile des ersten Zeichens ganz am Anfang in Byte 1. Die erste Zeile des zweiten Zeichens schließt sich daran an, die erste Zeile des dritten Zeichens wiederum daran usw. D.h., daß die ersten 256 Byte des Font-Speichers der Reihe nach aus den ersten Zeilen der 256 Zeichen bestehen. Danach kommt ein Block von wieder 256 Byte, der nacheinander die jeweils zweite Zeile der Zeichen enthält. Der nächste 256-Byte-Block enthält der Reihe nach die jeweils dritte Zeile der Zeichen usw. Bei 16 Zeilen je Zeichen macht das eine Font-Speichergröße von genau 256*16 = 4096 Bytes aus.

Für die Disketten-TOS-Benutzer hier die Adressen des 8*8- und des 8*16-Fonts sowie die üblichen Adressen der AES-Images:

8*8 - Font	=	101884
8*16- Font	=	104536
Füllmuster	=	97746

Die Füllmuster bestehen aus hintereinander liegenden 16-Byte-Blöcken, die die jeweils 1 Word breiten (16 Bit = 2 Byte) und 8 Zeilen hohen Muster-Images enthalten.

Desk-Icons = 125138

Die Desktop-Icons sind folgendermaßen organisiert:

An der angegebenen Adresse beginnt die Maske des Disk-Symbols. Die Maske umfaßt immer 128 Byte (4 Byte pro Zeile * 32 Zeilen = 128 Byte). Daran schließen sich der Reihe nach an:

Disk-Symbol-Maske	(+0)
Disk-Symbol-Image	(+128)
Ordner-Symbol-Maske	(+256)
Ordner-Symbol-Image	(+384)
Müll-Symbol-Maske	(+512)
Müll-Symbol-Image	(+640)
PRG-Symbol-Maske	(+768)
PRG-Symbol-Image	(+896)
Datei-Symbol-Maske	(+1024)
Datei-Symbol-Image	(+1152)

Alert-Icons = 113578

Die Alert-Icons sind ebenso in 128er Abständen organisiert, nur daß hier drei Images aufeinanderfolgen, da die Alert-Icons über keine Maske verfügen.

Ausrufungszeichen	(+0)
Stop-Schild	(+128)
Fragezeichen	(+256)

Maus-Icons = 113962

Die Maus-Icons sind folgendermaßen organisiert:

Die ersten 10 Byte enthalten Angaben über die X-Koordinate des Aktionspunktes innerhalb des Icons (Word 1 = 0 bis 15), die Y-Koordinate des Aktionspunktes (Word 2 = 0 - 15) und über Normal- oder XOR-Darstellung (Word 3 = 0 oder 1). Die Bedeutung von Word 4 und 5 ist mir nicht bekannt. An diesen Block schließt sich jeweils die Mausmaske (2*16=32 Byte) und

daran das Maus-Image (32 Byte) an. Im ersten 74-Byte-Block liegt das Maus-Icon für 'DEFMOUSE 0'. Daran schließen sich sieben weitere 74-Byte-Blöcke für 'DEFMOUSE 1-7' an, die ebenso aufgebaut sind wie der erste Block. Der gesamte Mausblock ist also 74*8 = 592 Bytes lang.

Weitere Adressen kann ich den Disk-TOS-Besitzern nicht bieten, da ich genug damit zu tun hatte, die ROM-TOS-Adressen herauszufinden.

Für die ROM-TOS-Anwender gilt dieselbe Block-Organisation für Desktop-, Alert- und Maus-Icons wie oben beschrieben, nur daß die Adressen hier an anderer Stelle liegen. Um die richtigen Adressen zu finden, müssen Sie von den oben genannten Startadressen folgende Offsets abziehen:

Desktop-Icons	=	-68754
Alert-Icons	=	-67766
Maus-Icons	=	-67766

Der Zugriff auf die Fonts und Füllmuster ist Ihnen aus oben genannten Gründen verwehrt. Anschauen können Sie sich diese jedoch, indem Sie mit dem Memory-Spion im Speicher ganz weit nach hinten fahren und langsam das ROM durchscrollen.

Als Ausgleich für diesen Mangel folgen nun einige hochinteressante RAM-Adressen:

2482 Media-Change-Flag setzen (1 Word)

Dpoke 2482,65535 bewirkt, daß das System die Diskette als gewechselt ansieht und z.B. beim nächsten Fileselect-Box-Aufruf das aktuelle Directory neu lädt.

3582 Maustastenstatus bestimmen (1 Byte)

Poke 3582,248 = keine Maustaste gedrückt
Poke 3582,249 = rechte Maustaste gedrückt
Poke 3582,250 = linke Maustaste gedrückt
Poke 3582,251 = beide Maustasten gedrückt

Beispiel:

```
' Programm: BSP1_5_1.BAS

Do
  Poke 3582,250              ! linke Maustaste ist gedrückt!
  If Mousek=1                ! rechte Maustaste gedrückt?
    Circle Mousex,Mousey,4 ! Ja, Kreis zeichnen
  Endif
Loop
```

3583 Mausbewegung links/rechts (1 Byte)

Peek(3583) = Maus nach links (Bit 0-2 an) oder rechts (Bit 5-7 an) bewegt?

3584 Mausbewegung hoch/runter (1 Byte)

Peek(3584) = Maus nach unten (Bit 0-2 an) oder oben (Bit 5-7 an) bewegt?

Die Bits geben hier an, wie schnell die Maus bewegt wurde:

Bit 0 bzw. Bit 7 gesetzt = langsam
Bit 2 bzw. Bit 5 gesetzt = schnell

Beispiel:

```
' Programm: BSP1_5_2.BAS

On Break Gosub Ende                 ! Abbruch kontrollieren
Deftext ,,,6                        !  kleinere Schriftgröße
Openw 0                             !  Window 0 auf
Do
  A=Peek(3583)                      !   rechts/links-Bewegung abfragen
  B=Peek(3584)                      !   hoch/runter -Bewegung abfragen
  If (A And 7) And A<15             !   untere 3 Bits (3583) abfragen
    Print "rechts"''A''             !    Bewegung anzeigen
  Endif
  If A And 224                      !   obere 3 Bits (3583) abfragen
    Print "links"''256-A''          !    Bewegung anzeigen
  Endif
```

```
  If (B And 7) And B<15                  !  untere 3 Bits (3584) abfragen
    Print "unten"''B''                   !   Bewegung anzeigen
  Endif
  If B And 224                           !  obere 3 Bits (3584) abfragen
    Print "oben"''256-B''                !   Bewegung anzeigen
  Endif
  Print
  Exit If (B<248 And B>7) Or (A<248 And A>7)!schnelle Bewegung = Abbruch
Loop
Ende
Procedure Ende
  Closew 0                               ! Window 0 schließen
  Edit
Return
```

3592 Joystick-Port 0 abfragen (1 Byte)

3593 Joystick-Port 1 abfragen (1 Byte)

Mit diesen beiden Adressen kann der Status beider Joystickports ermittelt werden. Während 3593 permanent Auskunft über den Port 1 gibt, muß, um Port 0 abfragen zu können, erst der Joystick-Modus im IKB eingeschaltet werden. Das geschieht mit 'Out 4,20'. Ab jetzt werden die Signale am Port 0 nicht mehr als Maus-(trigger)-Bewegungen, sondern als Joystick-Aktionen interpretiert. Soll die Maus wieder in Aktion treten, ist durch 'Out 4,8' der Mausmodus wieder einzuschalten.

Beispiel (Joystick in Port 0!):

```
' Programm: BSP1_5_3.BAS

On Break Gosub Ende
Out 4,20
Do
  A%=Peek(3592)
  If A% And 1
    Print "hoch";
  Endif
```

```
    If A% And 2
      Print "runter";
    Endif
    If A% And 4
      Print "links";
    Endif
    If A% And 8
      Print "rechts";
    Endif
    If A% And 128
      Print "Feuer";
    Endif
  Loop
  Procedure Ende
    Out 4,8
    Edit
  Return
```

3611 Sondertasten-Status fragen/setzen (1 Byte)

Dieses Byte erfüllt den gleichen Zweck wie die BIOS-Funktion 11 und wird auch von dieser benutzt.

9952 Maus-X-Koordinate setzen (1 Word)

9954 Maus-Y-Koordinate setzen (1 Word)

Durch Dpokes in diese beiden Adressen können Sie die Position der Maus auf dem Bildschirm bestimmen, ohne die Maus bewegen zu müssen. Im Memory-Spion haben Sie bereits eine Einsatz-Möglichkeit kennengelernt. Man kann durch die gezielte Manipulation dieser Adressen eine Skalierung der Mausposition erreichen. D.h., daß bestimmte Bereiche des Bildschirms nicht mit der Maus 'betreten' werden können. Sie müssen dabei allerdings peinlichst darauf achten, daß der ge'dpoke'te Wert nie unter 0 oder über der maximalen Randkoordinate rechts oder unten liegt. Das gibt einen glatten Absturz. Wollen Sie die Mauskoordinate durch 'Dpeek' aus diesen Adressen erfahren,

können Sie absolut sicher davon ausgehen, daß die gelieferten Werte immer auf die linke, obere Bildschirmecke bezogen sind.

Beispiel:

```
' Programm: BSP1_5_4.BAS

Print "Maus mit Schwung in das Feld bringen!"
Box 200,80,440,320
Dpoke 9952,10
Dpoke 9954,10
Do
  If Dpeek(9952)<320 And Dpeek(9952)>200
    Dpoke 9952,200
  Endif
  If Dpeek(9954)<200 And Dpeek(9954)>80
    Dpoke 9954,80
  Endif
  If Dpeek(9952)>206 And Dpeek(9954)>86 Or Mousek
    Edit
  Endif
Loop
```

10206 letzte Mausaktion ermitteln (1 Byte)

In dieser Adresse speichert das System die zuletzt stattgefundene Mausaktion. Wurde also z.B. zuletzt die rechte Maustaste gedrückt, kann das durch 'Dpeek(10206) auch nach einiger Zeit noch ermittelt werden. In diesem Fall ist das Bit 7, bzw. 'Dpeek(10206) And 128' liefert 'True'.

Liefert 'Dpeek(10206) And 64' True, so ist Bit 6 an, und es wurde zuletzt die linke Maustaste gedrückt. Mit 'Dpeek(10206) And 32' kann zusätzlich noch erfahren werden, ob seit der letzten Tastenbetätigung die Maus bewegt wurde oder nicht.

Wurde sie bewegt, liefert dieses 'Dpeek' ein 'True'. Liefert es ein 'False', weiß man, daß die Maus in der Zwischenzeit nicht bewegt wurde.

10226 Byteposition des Mauszeigers (1 Long)

Hiermit kann das absolute Byte ermittelt werden, auf dem sich der Mauszeiger zur Zeit innerhalb des Bildschirmspeichers befindet.

10232 Maushintergrund (16 Longs)

Das System muß ständig den Maushintergrund zwischenspeichern, um den durch die Maus verdeckten Bereich restaurieren zu können. Dies geschieht in den Adresse 10232 - 10295. Von dort holt sich das System im Interrupt diese 16 Zeilen (je 32 Bit) und flickt damit den Bildschirmspeicher, sobald die Maus bewegt wurde. Schreibt man nun in diese Adressen etwas Willkürliches, wird anschließend also damit die Screen repariert.

Beispiel:

```
' PRogramm: BSP1_5_5.BAS

Repeat
  Lpoke 10232+Random(8)*4,Random(2^31-1)
Until Mousek
```

Zum Schluß noch ein kleiner Trick zum Interpreter.

Manchmal ist es sehr störend, daß bei jedem Programmstart der Bildschirm gelöscht wird. Mit der folgenden Routine können Sie dieses Anfangs-CLS aus- oder auch wieder einschalten.

```
' Programm: PROC1_5.BAS
'
Procedure Xcl(Flg%) ! Flg% = 0 => CLS aus / Flg% <> 0 => CLS an
  Local A$
  A$=Space$(3000)
  Bmove Basepage,Varptr(A$),3000
  If Flg%=0
    Poke Basepage+Instr(A$,Chr$(27)+"E"),Asc("H")
```

```
  Else
    Poke Basepage+Instr(A$,Chr$(27)+"H"),Asc("E")
  Endif
Return
```

Da ich mir nicht sicher bin, ob der betreffende Escape-Befehl in allen Interpreter-Versionen an derselben Position steht, werden zuerst die ersten 3000 Bytes hinter der Basepage nach der Sequenz durchsucht. Je nachdem, ob das CLS aus- oder eingeschaltet werden soll, wird dann entweder der Escape-Befehl "E" (ClearHome) durch "H" (Home) ersetzt, oder umgekehrt. Bei meiner 'upgedateten' V2.O-Version befindet sich das entscheidende Byte an Position 'Basepage+1149'.

1.6 Atari-gesicherte System-Adressen

Die Spezial-Adressen, die ich Ihnen oben vorgestellt habe, sind, wie bereits mehrfach erwähnt, nicht gesichert. D.h., daß sie durch verschiedene Umstände verschoben sein können. Zu diesen Umständen gehören vor allem residente Auto-Boot-Programme und verschiedene TOS-Versionen.

Wie von Atari verkündet, sollen die folgenden Adressen unveränderlich sein. Da sehr viele professionelle Programme sich diese Adressen zunutze machen (z.B. GFA-BASIC), waren seit der MEGA-ST-Ankündigung (neues ROM-TOS) einige Software-Fabrikanten in Aufruhr. Atari hat nun zugesichert, daß die Adressenlage innerhalb des Supervisor-Bereichs (Adressen 0 - 2047) unverändert bleibt.

In diesem Kapitel finden Sie nun eine Sammlung von 32 Adressen, die für das System sehr wichtige Funktionen haben. Ohne die Richtigkeit bzw. Brauchbarkeit dieser Adresseninhalte wäre es über kurz oder lang zum Absturz verurteilt. Das ist auch der Grund, warum dieser Bereich durch den Supervisor schreibgeschützt ist und man nur im Supervisormodus (Spoke, Sdpoke, Slpoke) diese Daten verändern kann.

Viele dieser Adressen sind für den BASIC-Programmierer uninteressant, da man ihre Modifikation nur in Assembler oder 'C' nutzen kann. Deshalb habe ich mir erlaubt, nur diese Auswahl anzubieten.

Zuerst folgt ein kleines Programm, das Ihnen die Bedeutung, Dig.Research-Kürzel, Adresse, den Inhalt und das Format der Daten anzeigt. Weiter läßt sich mit diesem Programm nichts anfangen. Am Ende des Programms befinden sich die Data-Zeilen, anhand derer ich die Einzelheiten erläutern werde.

```
' Programm: SYS_ADRS.BAS
'
' |''''''
```

```
    Endif
    If Mod$(I%)="W"                          !  Wordformat?
      Ofs%=J%*2                              !   Offset
      Inh%=Dpeek(Adr%(I%)+Ofs%)              !   Word-Inhalt auslesen
    Endif
    If Mod$(I%)="L"                          !  Longwordformat
      Ofs%=J%*4                              !   Offset
      Inh%=Lpeek(Adr%(I%)+Ofs%)              !   Long-Inhalt auslesen
    Endif
    Print At(57,Crslin);Adr%(I%)+Ofs%'''Inh%,''Mod$(I%) ! anzeigen
  Next J%
Next I%
Hidem
Line 300,0,300,366
Line 437,0,437,366
Line 497,0,497,366
Line 570,0,570,366
Graphmode 3
Repeat                                  !---!
  Mouse X,Y,K                               !
  Box 7,Y-14,632,Y                          !
  Box 8,Y-13,631,Y-1                        !
  Poke 3584,0                               !
  Repeat                                    !-- Lineal
    Key%=Asc(Inkey$)                        !
  Until Peek(3584) Or Mousek Or Key%        !
  Box 7,Y-14,632,Y                          !
  Box 8,Y-13,631,Y-1                        !
Until Mousek Or Key%                    !---!
Showm
Closew 0
Edit
Sys_adr:
Data Konfig.-Copy  d. Memory-Contr., memctrl ,&H424,1,W
```

Kopie des Konfigurationswertes im Memory-Controller:

```
Data Reset-Zulassung, resvalid ,&H426,1,L
```

Wird hier der Wert &H31415926 eingetragen, kann damit erreicht werden, daß bei einem Reset über den Reset-Vektor in Routinen gesprungen wird, die auf den Reset reagieren. Ohne genaueste Kenntnis des Vorgangs läßt sich damit leider nichts erreichen.

```
Data Reset-Vector, resvector ,&H42A,1,L
```

Dieses ist der eben genannte Reset-Vektor, der allerdings keinen Inhalt aufweist. Soll auf den Reset reagiert werden, wird hier die Adresse der reagierenden Routine eingetragen.

```
Data RAM-Physical-End, phystop ,&H42E,1,L
```

Wie der Name sagt, ist in dieser Adresse das Ende des physikalischen RAMs eingetragen.

```
Data Benutzer-Speicher-Start, _membot ,&H432,1,L
```

Die Anfangsadresse des Benutzerspeichers finden Sie hier.

```
Data Benutzer-Speicher-Ende, _memtop ,&H436,1,L
```

Und hier die Endadresse.

```
Data Read/Write-Seekrate, seekrate ,&H440,1,W
```

Ist Ihnen Ihre Floppy zu langsam? Wenn Sie in diese Adresse den Wert 2 Lpoken, wird sie etwas schneller. Damit wird die Geschwindigkeit der Schreib-/Lesekopf-Bewegung von einem Track zum nächsten bestimmt.

0 = 6 Millisek. / 1 = 12 Millisek.
2 = 2 Millisek. / 3 = 3 Millisek.

Der Default-Wert ist die 3.

```
Data Timer-Differenz, _timer_ms ,&H442,1,W
```

Diese Adresse zeigt Ihnen die Differenz zwischen zwei System-Timer-Aufrufen in Millisekunden.

```
Data Write-Verify on/off, _fverify ,&H444,1,W
```

Diese Speicherstelle bestimmt, ob bei Floppy-Schreibzugriffen automatisch ein Verify ausgeführt wird. Wenn Sie das Verify unterdrücken möchten (Floppy schreibt schneller), müssen Sie diese Adresse auf Null setzen. Jeder andere Wert schaltet das Verify wieder ein.

```
Data Boot-Laufwerksnummer, _bootdev ,&H446,1,W
```

Hier kann die Nummer des Laufwerks ermittelt werden, von dem das System gebootet wurde. Bei ROM-TOS-Maschinen ist das relativ uninteressant.

0 = A: / 1 = B:

```
Data PAL(50Hz) / NTSC(60Hz)-Modus, palmode ,&H448,1,W
```

Hier kann erfahren werden, ob das Sytem im 50-Hertz-PAL-Modus (<>0) oder im 60-Hertz-NTSC-Modus (=0) arbeitet. Veränderungen an dieser Adresse haben keinen Sinn, da der Modus nur während des Bootens installiert werden kann.

```
Data Neue Color-Auflösung, defshiftmod ,&H44A,1,W
```

Bei Umschaltung vom Monochrom- in den Farbmodus liefert diese Adresse dem System die gewünschte Farb-Auflösung. Ohne System-Kenntnisse kann dieses Adresse nur gelesen werden.

0 = LOWRES / 1 = MIDRES

Data Register-Copy d. Auflösung, sshiftmd ,&H44C,1,W

Auch diese Adresse kann nur gelesen werden. Sie liefert den gleichen Wert, der auch durch 'XBIOS(4)' erfahren werden kann.

0 = LOWRES / 1 = MIDRES / 2 = HIRES

Data Zeiger auf Logbase, _v_bas_ad ,&H44E,1,L

Diese Adresse liefert denselben Wert wie 'XBIOS(3)': einen Zeiger auf die Startadresse des logischen Bildschirms. Wollen Sie die Screen-Adressen verändern, können Sie dies am besten mit 'XBIOS(5).

Data VBL-Routine on/off, vblsem ,&H452,1,W

Durch Setzen einer Null in dieser Adresse können die in &H454 eingetragenen Vertikal-Blank-Interrupt-Routinen gesperrt werden. Dadurch wird dann z.B. auch die Abbruchfunktion des GFA-BASIC außer Kraft gesetzt. Durch Eintragen einer 1 wird die VBL-Routine wieder aktiviert.

Data Anzahl d. VBL-Routinen, nvbls ,&H454,1,W

In diesem Longword steht die Anzahl der auszuführenden Vertikal-Blank-Interrupt-Routinen. Als Default-Wert ist hier eine 8 (8 Routinen) eingetragen. Vom System wird nur eine VBL-Routine installiert. Die anderen 7 Routinen können frei bestimmt werden. Durch Heraufsetzen der Zahl können entsprechend viele Routinen angemeldet werden.

Data nvbls-Routinen-Liste, _vblqueue ,&H456,1,L

Dies ist ein Zeiger auf eine Liste von Adressen der in &H454 angemeldeten VBL-Routinen. Im Defaultzustand ist diese Liste 8 Longwords lang (8 eingetragene Routinen). Durch 'Lpeek(Lpeek(&H456))' erfahren Sie die Startadresse der ST-Standard-Routine. Das zweite Longword 'Lpeek(Lpeek(&456)+4)' zeigt sehr wahrscheinlich auf eine Adresse im BASIC-Interpreter, wo sich eine vom BASIC eingetragene Routine befindet. Sollte das nicht der Fall sein, wurden vorher durch ein Auto-

Boot-Programm eine oder mehrere VBL-Routinen angemeldet, und der Eintrag der BASIC-Routine verschiebt sich um dementsprechend viele Longwords in der Liste. Die nächsten Einträge hinter dem BASIC-Eintrag müßten dann null sein. Dort kann man dann die Adressen seiner eigenen Interrupt-Routinen eintragen und damit erreichen, daß diese Routinen dann vom ST treu und brav bei jedem VBL-Interrupt ausgeführt werden.

Sollen mehr als 8 VBL-Routinen ausgeführt werden, ist die Liste entsprechend zu erweiteren, in einen ausreichend großen Speicherbereich zu verlegen, in &H456 die neue Tabellen-Adresse und in &H454 die neue Anzahl einzutragen.

Ohne fundierte Assembler-Kenntnisse ist hier jedoch nicht viel auszurichten. Übrigens können Sie die Abbruchfunktion des BASIC total ausschalten, indem Sie in das vierte Byte der BASIC-Interrupt-Routine einfach eine Null schreiben. Wollen Sie auch noch den Direkt-Modus im Interpreter ausschalten, schreiben Sie in das fünfte Byte einfach ebenfalls eine Null. Nur darf danach kein 'On Break Cont'-Befehl mehr ausgeführt werden, da sonst die Break-Umschaltung im Interpreter durcheinander gerät. Wollen Sie beides wieder einschalten, poken Sie in das vierte Byte eine 27 und in das fünfte Byte eine 2.

Die BASIC-Interrupt-Routine kann man in der Liste dadurch identifizieren, daß sie ca. 26 KByte über der BASIC-Basepage, also innerhalb des Interpreters liegen muß.

Data Neue Farbpalette installieren, colorptr ,&H45A,1,L

Hier kann die Adresse einer 16 Words umfassenden Farbpalette eingetragen werden. Diese Palette wird dann beim nächsten VBL-Interrupt geladen, sofern der VBL-Interrupt nicht gesperrt ist. Damit keine Palette geladen wird, ist der Wert 0 einzutragen.

Data Video-RAM-Adresse installieren, screenpt ,&H45E,1,L

Diese Adresse wird als Zeiger auf die Startadresse des logischen Bildschirmspeichers interpretiert. Das neue Video-RAM wird dann beim nächsten VBL-Interrupt installiert. Soll nicht ständig das Video-RAM installiert werden, muß diese Adresse natürlich Null enthalten.

`Data Attribut-Vektor für Console, conterm ,&H484,1,B`

Die untersten 4 Bit dieses Bytes bestimmen, ob:

- der Tastatur-Klick ein (Bit 0=1) oder aus (Bit 0=0) ist,
- der Tastatur-Repeat ein (Bit 1=1) oder aus (Bit 1=0) ist oder
- die BIOS-Funktion 2 (Conin) in den obersten 8 Bit den Umschalttastenstatus liefert (Bit 3=1).

Das Bit 2 bestimmt zusätzlich die Wirkung von <Control><G> auf die Soundverarbeitung (an/aus?). Welche Wirkung das sein soll, konnte ich nicht herausfinden.

`Data Memory-Descriptor, themd ,&H48E,4,L`

Mit der BIOS-Funktion 0 wird die Adresse des Memory-Parameter-Blocks bestimmt. Als Rückgabewert erhält man die Adresse dieses Memory Descriptors. Weiteres s. unter BIOS(0).

`Data Zeiger auf Register-Save-Area, savptr ,&H4A2,1,L`

Man hat hierdurch eine Möglichkeit, sich Einblick in den Stand der Prozessor-Register zu verschaffen. Bei jedem BIOS-Aufruf werden die Register-Inhalte in einem bestimmten Bereich gesichert. Der Inhalt dieser Adresse stellt nun einen Zeiger auf die Anfangsadresse dieses Bereichs dar.

`Data Anzahl angeschlossener Floppies, _nflops ,&H4A6,1,W`

Diese Adresse liefert einen Wert, der die Anzahl der angeschlossenen Laufwerke angibt. Genau wie die BIOS-Funktion 10 wird immer davon ausgegangen, daß mindestens zwei Laufwerke (A und B) angeschlossen sind.

`Data 200-Hz-Systemcounter, _hz_200 ,&H4BA,1,L`

Dieses ist der System-Timer, der von dem BASIC-Befehl 'Timer' benutzt wird.

Beispiel: Print Timer'"Lpeek(&H4BA)

```
Data Bit-Vektor f. angeschl. Laufwerke, _drvbits ,&H4C2,1,L
```

Aus dieser Adresse kann ermittelt werden, welche Laufwerke zur Zeit angeschlossen sind. Es handelt sich hier um einen Bit-Vektor. Ist Bit 0 gesetzt, heißt das, daß Floppy A: angeschlossen ist. Bit 1 steht für Floppy B:, Bit 2 für C: usw. Auch hier geht das System davon aus, daß generell mindesten Drive A: und B: angeschlossen sind.

```
Data 1024-Byte-DiskPuffer, _dskbufp ,&H4C6,1,L
```

Um nicht bei jedem Diskettenzugriff die aktuellen Disketten-Attribute von Disk holen zu müssen, richtet sich das System einen Disk-Puffer ein, in welchem der Boot-Sektor abgelegt wird. Dieser 1024 Byte große Puffer beginnt an der hier eingetragenen Adresse.

```
Data Standard-VBL-Routine, _vbl_list ,&H4CE,8,L
```

Dies ist die VBL-Routinenliste, deren Adresse durch 'Lpeek(&H456)' erfragt werden kann.

In ihr sind der Reihe nach alle VBL-Routinen eingetragen, welche bei jedem VBL-Interrupt ausgeführt werden sollen. Wie oben bereits erwähnt, können auch mehr als acht Routinen angemeldet werden, wenn dieser entsprechend verlängerte Vektor an eine ausreichend große Speicherstelle verlegt wird, die neue Anzahl in &H454 und der neue Zeiger in &H456 eingetragen wird.

```
Data Hardcopy-Flag on/off, _dumpflg ,&H4EE,1,W
```

Eine 1 in dieser Adresse zeigt an, daß gerade eine Hardcopy gefertigt wird. Es ist möglich, allein durch Setzen einer 0 in dieser Adresse eine Hardcopy auszulösen. Der Default-Inhalt dieser Adresse ist 65535. Setzt man hier eine 1, wird mit jedem <Alternate><Help> der Adresseninhalt um 1 erhöht. Eine Hardcopy durch <Alternate><Help> wird jedoch nur ausgeführt, wenn vorher der Wert 65535 in dieser Adresse enthalten ist. D.h. also, daß man durch Eintragen eines Wertes > 0 die Hardcopy-Funktion <Alternate><Help> unterdrücken kann. Sie ist erst

wieder in Takt, wenn der Wert 65535 eingetragen wurde. Hat man die Hardcopy-Funktion ausgeschaltet, kann man außerdem feststellen, ob vom Benutzer in der Zwischenzeit <Alternate><Help> gedrückt wurde, weil ja bei jedem Druck auf diese Tastenkombination der Zähler um 1 erhöht wird. Ist der aktuelle Wert also größer als der von Ihnen eingetragene, wurde in der Zwischenzeit versucht, eine Hardcopy auszuführen.

```
Data Printer-Time-Out-Abort-Flag, _prtabt ,&H4F0,1,W
```

Wozu dieses System-Flag gut ist, konnte ich nicht in Erfahrung bringen.

```
Data Beginn des Betriebssystems, _sysbase ,&H4F2,1,L
```

Diese Adresse enthält einen Zeiger auf die Startadresse des Betriebssystems.

```
Data Ende des Betriebssystems, end_os ,&H4FA,1,L
```

Hier ist das Ende des Betriebssystems im RAM eingetragen. Das Betriebssystem benötigt auch bei einem ROM-TOS einen gewissen Bereich des RAMs für sich. Das Ende dieses RAM-Bereichs ist hier gemeint.

```
Data Beginn des AES, exec_os ,&H4FE,1,L
```

Hier finden Sie einen Zeiger auf die Startadresse des AES innerhalb des Betriebssystems.

2. Das Floppy-ABC

Inhalt und Aufbau der alles entscheidenden Disketten ist wohl den meisten Lesern ein Buch mit den berühmten sieben Siegeln.

Alles entscheidend deshalb, weil unser Super-ST ohne diese kleinen schwarzen 'Schwabbel-Scheiben' (engl.: Floppy-Disk) nun mal nicht auskommt.

So eine Diskette sieht eigentlich nicht besonders exclusiv aus. Es handelt sich um eine dünne Kunststoffscheibe, die mit einer magnetischen Oberfläche beschichtet ist. Die Daten werden grundsätzlich in der gleichen Verfahrensweise in diese magnetische Schicht "eingraviert", wie wir es von Magnet-Tonbändern her kennen. Der Unterschied ist der, daß bei einem Tonband die Informationen analog zu der dem Tonkopf zu- geführten Spannung übertragen werden, d.h. daß hier je nach Qualität des Gerätes und des Bandes eine sehr große Frequenzspanne vorliegen kann. Bei einer Diskette werden ausschließlich digitale Informationen verarbeitet. Der Tonkopf überträgt also nur zwei Frequenzen. Die zu übertragenden Daten werden bitweise interpretiert, und je nachdem, ob ein Bit gesetzt oder leer ist, wird entweder ein hoher oder ein niedriger Ton gesendet. Die Frequenz dieser beiden unterschiedlichen Töne hinterläßt dann auf der Magnetschicht ganz charakteristische Spuren.

Da sich die Diskette während des Schreibens und Lesens ständig um ihre Achse dreht, ist der Gedanke nicht fern, sie in konzentrische Kreise einzuteilen.

Womit wir auch schon bei den mysteriösen Tracks (engl.: Spuren) angekommen sind.

Sind Sie im Besitz eines besseren Formatier-Programms, wird Ihnen schon aufgefallen sein, daß die Anzahl an Tracks, die sich auf einer Diskette befinden, auch bei gleichem Diskettentyp unterschiedlich sein kann. Man hat dort also die Wahl, ob man die Diskette in 40, 80, 81 oder 82 Tracks (je Diskettenseite)

einteilen möchte. Das bedeutet nichts anderes, als daß nun der zur Verfügung stehende Teil des Diskettenradius' physikalisch in die gewünschte Track-Anzahl unterteilt wird.

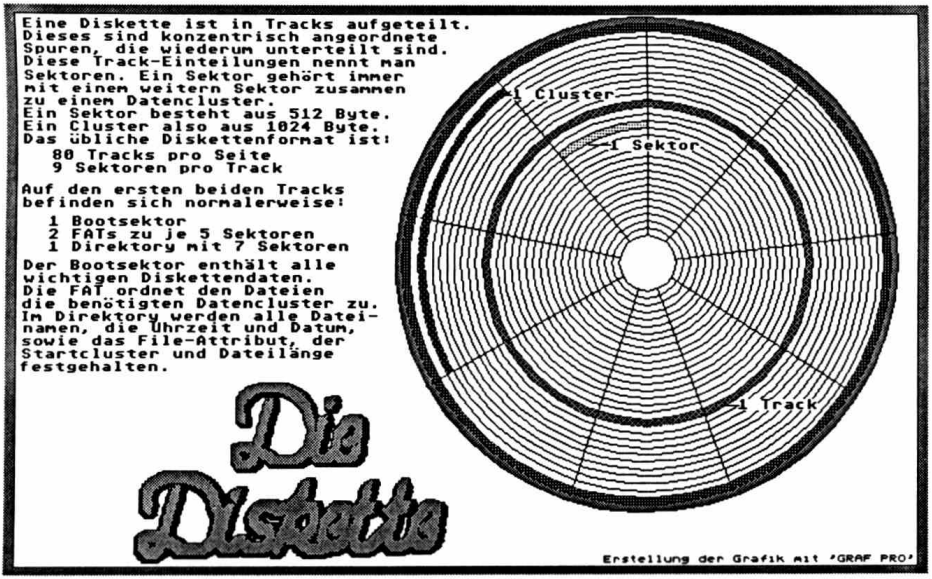


Abb. 4: **Die Diskette**

Weiterhin wird Ihnen vielleicht aufgefallen sein, daß auch die Anzahl der Sektoren variabel sein kann. Wird z.B. eine einseitige Diskette mit 82 Tracks zu je 10 Sektoren formatiert, stehen Ihnen 82*10 = 820 Sektoren zu je 512 Byte (820*512 = 419840 Byte) zur Verfügung. Da man auf diese Weise ca. 60 Kilobyte pro Diskettenseite mehr an Speicherplatz erhält, wird dieses Diskettenformat recht häufig verwendet.

Aus einem ganz bestimmten Grund erweist es sich jedoch oft als nachteilig, seine Diskette derart platzsparend einzurichten. Das Betriebssystem des ST ist in seinen Kopier- und Formatier-Utilities auf das großzügigere 720-Sektoren-Format ausgerichtet, und man hat so keine Möglichkeit, ohne spezielle Kopierpro-

gramme seine Disketten zu vervielfältigen. Sie werden also üblicherweise Disketten vorfinden, die mit 80 Tracks zu je 9 Sektoren (80*9 = 720 Sektoren) ausgestattet sind.

Wer nun schnell mitgerechnet hat, kommt auf eine Bytezahl von 720*512 = 368640 Byte je Diskettenseite. Nach dem Formatieren mit dem Desktop-Formatter werden Ihnen jedoch nur 357376 als frei gemeldet. Das liegt daran, daß das System sich einen hübschen Batzen von 22 Sektoren zur Diskettenverwaltung, also für Bootsektor, File-Allocation-Tables (engl.: Datei-Zuteilungs-Tabelle) und Directories abkneift. Es bleiben also nur noch 698 Sektoren (698*512 = 357376 Byte) je Diskettenseite übrig. Ein erfreulicherweise für die meisten Zwecke völlig ausreichender Speicherplatz.

Nun sind gleich drei Begriffe gefallen, deren Erklärung zum Verständnis der Disketten-Organisation unumgänglich ist.

Was sind eigentlich Sektoren? Sektoren sind die kleinste Einteilungsgröße auf der Diskette. Sie bestehen generell aus 512 Byte. Während ein Track eher eine technische Einheit ist, ist der Sektor zur Verwaltung der Diskette notwendig.

Boot-Sektor

Jede Diskette verfügt über einen solchen. Er ist generell auf Sektor 0 (Seite 0, Track 0) einer Diskette zu finden. Nach erfolgter Formatierung einer Diskette werden hier grundlegende Informationen zur Disketten-Organisation festgeschrieben (siehe unter XBIOS(18)).

Bei genauester Kenntnis der Materie ist es auch möglich, ihn nachträglich zu modifizieren, um so z.B. versteckte Tracks zu erzeugen.

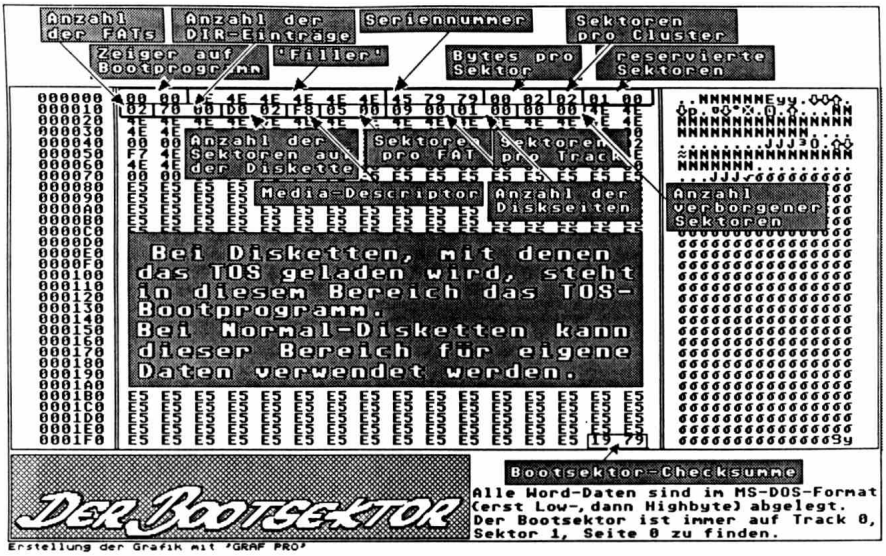


Abb. 5: Der Bootsektor

File-Allocation-Table

Wie der Name schon sagt, wird hier den Dateien etwas zugewiesen. Wird eine Datei eingerichtet, werden in den FAT-Sektoren (üblich 2*5) die Sektoren eingetragen, die von dieser Datei belegt werden. Eine FAT muß immer so groß sein, daß für jeden Sektor der Diskette eineinhalb Byte (2*6 Bit) zur Verfügung stehen. Die FAT beginnt immer mit der Drei-Byte-Konstante &HF7FFFF. Üblicherweise werden pro Diskette zwei FAT's angelegt, wovon die zweite direkt auf die erste folgt. Wurde z.B. eine einseitige Disk mit 80 Tracks zu je 10 Sektoren formatiert, müssen die FAT's mindestens 80*10*1.5/512 = 3 Sektoren groß sein. Da die erste FAT üblicherweise auf Sektor 1 in Track 0 der Seite 0 einer Diskette liegt, würde die zweite in diesem Fall also auf Sektor 4 folgen.

Directory

Ein Directory (engl.: Adressbuch) beinhaltet üblicherweise die Namen aller Dateien, die jemals (seit der letzten Formatierung!) auf dieser Diskette angelegt wurden. Es sei denn, der Dateiname wurde per Monitor komplett gelöscht.

Wurden Dateien gelöscht (z.B. Desktop-Papierkorb oder Kill-Funktion), wird der erste Buchstabe des Dateinamens durch das SIGMA-Zeichen (&HE5E5) ersetzt und der FAT-Eintrag gelöscht. Wurden Ordner angelegt, bestehen außer dem Haupt-Verzeichnis noch soviele Unterverzeichnisse, wie Ordner vorhanden sind. Im Anschluß an die jeweiligen Datei- und Ordnernamen enthält ein Directory-Eintrag zusätzliche Informationen über Datum und Uhrzeit der Datei-Erstellung, über Datei-Attribute sowie über Dateilänge und Startcluster.

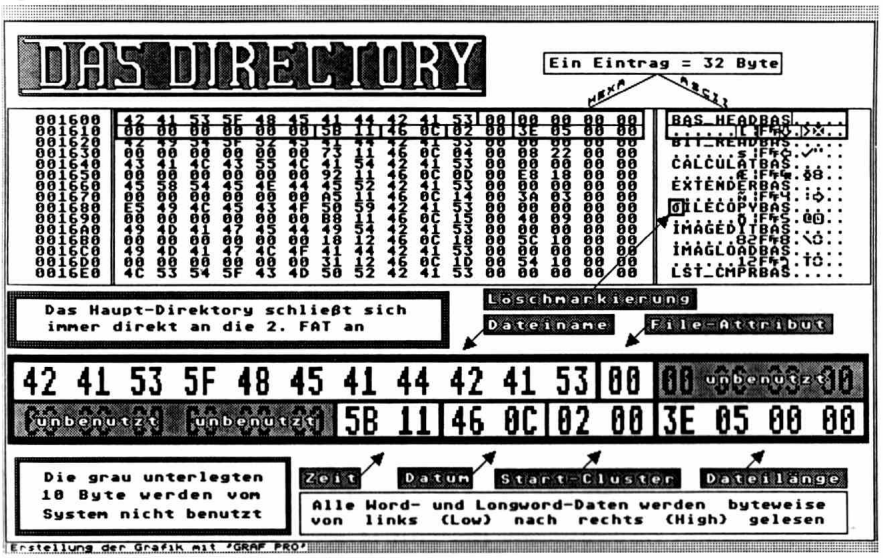


Abb. 6: Das Directory

Um nun handfestes Material zur Floppy-Lektion anbieten zu können, wollte ich zuerst nur ein paar kleine nützliche Routinen

entwickeln. Daraus hat sich dann frei nach dem Schneeball-Prinzip ein fast kompletter Disketten-Monitor entwickelt.

Beim Entwurf dieses Disketten-Monitors bin ich grundlegend davon ausgegangen, daß er seinen Zweck hauptsächlich darin hat, Disketten-Inhalte erforschen zu können. Im Lauf der Programm-Entwicklung sind mir dann allerhand weitere sinnvolle Aufgaben eines Monitors eingefallen, und ich habe versucht, einige davon zu verwirklichen.

Eine davon war das File-Recover, d.h. die Möglichkeit, gelöschte Dateien wieder brauchbar zu restaurieren. Leider muß ich gestehen, daß mir diese Funktion nicht 100%ig geglückt ist. Durch Umstände, die mir manchmal als sehr rätselhaft erschienen, gelingt es nicht immer, alle zu einer Datei gehörigen Cluster wiederzufinden. Wenn doch, kann es trotzdem noch passieren, daß trotz korrekter Eintragung in die File-Allocation-Tabelle Fehler auftreten. Dazu gehört vor allem, daß nach erfolgter Datei-Restauration das System mit der veränderten FAT nicht mehr zurechtkommt. Dabei habe ich die Erfahrung gemacht, daß dieser Mißstand behoben werden kann, indem kurzfristig eine andere (!) Diskette eingelegt wird (vorher <UNDO>-Taste drücken) und dann wieder die Disk, auf welcher sich das restaurierte File befindet. Aus mir unbekannten Gründen gibt es allerdings für das Auftreten dieses Fehlers keine Regel. Manchmal funktioniert es tadellos und manchmal eben nicht. Sollte nach einem File-Restore die Disketten-Verwaltung völlig durcheinander sein, hilft nur noch der RESET-Knopf. Die restaurierte Datei müßte danach allerdings wieder auf dem Desktop (mit dem Anfangsbuchstaben 'X') erscheinen. In selteneren Fällen wird dann beim Listen der Datei immer noch auf falsche Cluster zugegriffen. Dann bleibt nur noch der Versuch, die Datei auf eine andere Diskette oder auf dieselbe Diskette mit geändertem Namen zu kopieren und die Ursprungsdatei zu löschen. Sollte das immer noch nicht helfen, ist die Datei nicht mehr zu retten. Hier bleibt also ein Spielraum für all jene, die sich intensiv mit der Datei-Verwaltung des Systems auseiandersetzen wollen.

Sollte jemandem das Kunststück gelingen, den Restaurations-Algorithmus 100%ig in den Griff zu bekommen, kann er sich rühmen, eine der schwierigsten Aufgaben, die das Betriebssystem des Atari ST zu bieten hat, gelöst zu haben. Ich wünsche viel Erfolg und vor allem Geduld dabei.

Dieser Monitor ist der Einfachheit halber zum größten Teil auf ein einseitiges Laufwerk ausgerichtet. Ein Monitor, der sowohl ein- als auch zweiseitige Disketten lesen und verarbeiten kann, wäre erheblich umfangreicher.

Hier im Buch finden Sie nur einige der wichtigsten Prozeduren dieses Programms bzw. die wichtigsten Teile daraus erläutert. Das gesamte, komplett kommentierte Listing finden Sie unter 'GFA_DMON.BAS' auf der Diskette.

In den hier aufgeführten Prozedurteilen habe ich immer dort, wo im Programm Zeilen auftauchen, die für die Erläuterung nicht so wichtig sind, untereinanderstehende Punkte verwendet.

```
' Programm: GFA_DMON.BAS
'
' |''''''
```

```
Procedure Disktyp(Sec1%,Spt1%,Sid1%,Spf1%,Fat1%,Dlen1%,D_at1%)
  Local Bt%,Aa$,Bpb%
  @Secmod(2,0,1,0,*Aa$,"")                  !Bootsektor laden
  Bt%=Varptr(Aa$)                           !Puffer-Startadresse
  Sec2%=Peek(Bt%+20)*256+Peek(Bt%+19)       !Sektoren auf Disk
  Spt2%=Peek(Bt%+25)*256+Peek(Bt%+24)       !Sektoren pro Track
  Spf2%=Peek(Bt%+23)*256+Peek(Bt%+22)       !Sektoren pro FAT
  Sid2%=Peek(Bt%+27)*256+Peek(Bt%+26)       !SD oder DD (1-2) ?
```

Die Bedeutung dieser Daten finden Sie unter 'XBIOS(18)' beschrieben. An der verkehrt gelesenen Datenfolge (z.B. erst Byte 20, dann Byte 19) ist das MS-DOS-Wordformat zu erkennen.

```
  Bpb%=Bios(7,0)                            !Bios-Parameter-Block (BPB)
  '                                         !   ( s. unter BIOS(7) )
  If Bpb%=0                                 !Bios-Funktionsfehler
    Alert 1,"Fehler im |BIOS-Parameter-Block!",1," Aha ",D.b%
    D.len%=7                                ! Directorylänge =7Sektoren
    F.at2%=6                                ! 2.FAT Startsektor=6.Sektor
    D.at1%=18                               ! 1. Datensektor=18.sektor
  Else                                      !Bios-Parameter-Block OK !
    D.len%=Dpeek(Bpb%+6)                    ! aktuelle Directorylänge
    F.at2%=Dpeek(Bpb%+10)                   ! aktueller 2.FAT-Startsektor
    D.at1%=Dpeek(Bpb%+12)                   ! aktueller 1.Datensektor
    '                                       !    (s. unter BIOS(7) )
  Endif
```

Die folgende 'If'-Abfrage stellt fest, ob im Bootsektor überhaupt brauchbare Angaben über Sektorenanzahl, Sektoren pro Track und Sektoren pro FAT zu finden sind. Sind sie das nicht, werden diese Daten hier 'simuliert', damit die Diskette überhaupt lesbar ist, da das Programm sich in der Sektorenauswahl nach diesen Daten richtet.

```
  If Sec2%=0 Or Spt2%=0 Or Spf2%=0          !Grund-Daten im Bootsektor Null?
    Alert 1,"Bootsektor defekt !?!?",1," Aha ",D.b%
    Spf2%=5                                 !Sektoren pro FAT=5
    Spt2%=9                                 !Sektoren pro Track=9
    Sec2%=720*Sid2%                         !Sektorenanzahl 720(SD)/1440(DD)
  Endif
```

```
  If Sektor%>Sec2%-1                    !aktueller Sektorzähler
    '                                   !     größer als Sektorenanzahl
    Sektor%=0                           ! Zähler auf Null
  Endif
  *Sec1%=Sec2%                          ! --------|
  *Spt1%=Spt2%                                    !
  *Spf1%=Spf2%                                    ! -- Daten an Haupt-
  *Sid1%=Sid2%                                    !    Programm über-
  *Dlen1%=D.len%                                  !    geben.
  *Fat1%=F.at2%                                   !
  *D_at1%=D.at1%                        !---------|
Return
```

Die nun folgende Prozedur ist mit Abstand die wichtigste von allen, da ohne sie keine Sektoren von der Diskette gelesen bzw. auf diese geschrieben werden könnten.

Für diese wichtige Aufgabe wurde hier die BIOS-Funktion 4 verwendet, da sie im Gegensatz zu den XBIOS-Funktionen 8 und 9 nicht trackrelativ arbeitet, sondern sich auf die logischen Sektoren der Diskette bezieht, und außerdem die Menge der Sektoren, die mit nur einem Funktionsaufruf gelesen oder geschrieben werden können, nur durch die maximale Anzahl der Sektoren beschränkt ist. Man kann also über die Trackgrenzen hinweg lesen und schreiben.

Die Parameter, die diese Prozedur benötigt, sind recht umfangreich:

M%=Modus — 0 = Sektor(en) lesen (evtl. Disk-Wechsel feststellen)

1 = Sektor(en) schreiben (evtl. Disk-Wechsel feststellen)

2 = Sektor(en) lesen (Disk-Wechsel ignorieren)

3 = Sektor(en) schreiben (Disk-Wechsel ignorieren)

S%=Sektor — Nummer des ersten Sektors

A%=Anzahl	Anzahl der zu lesenden bzw. zu schreibenden Sektoren
D%=Laufwerk	Nummer des bezogenen Laufwerks
B%=Pufferadresse	bei M=0 oder M=2 (Lese-Puffer)
St$=Puffer	bei M=1 oder M=3 (Schreib-Puffer)

Am Anfang der Prozedur werden der Moduswert auf den Bereich 0-3 und die maximal gleichzeitig lesbare Sektorenanzahl auf den Bereich 0-9 reduziert, um unvorhersehbare Lese- und Schreibfehler zu vermeiden.

```
Procedure Secmod(M%,S%,A%,D%,B%,St$)
  Local Puffer$,Bk%,Al$
  M%=M% Mod 4                        ! Modus-Wert nur im Bereich 0 - 3
  A%=A% Mod 10                       ! Sektorenanzahl nur im Bereich 0 - 9
  If M%=0 Or M%=2                    ! Sektor(en) lesen?
    Puffer$=Space$(512*A%)           ! Platz schaffen für Lese-Daten
  Else                               ! Sektor(en) schreiben!
    Puffer$=St$                      ! Übergabe-String in Puffer setzen
    If S%=0                          ! Wenn der Bootsektor geschrieben
      '                              ! werden soll,
      Bmove Varptr(Puffer$),Lpeek(&H4C6),512 ! dann System-Disk-Puffer
    Endif                            !                 auch erneuern.
  Endif
  Bk%=Bios(4,M%,L:Varptr(Puffer$),A%,S%,D%) ! Funktion ausführen
  If Bk%<>0                          ! Bios-Funktions-Fehler aufgetreten?
    Al$="  ERROR # "+Str$(Bk%)+"|(s. GFA-Literatur)"
    Alert 1,Al$,1,"RETURN",Bk%
  Else                               ! Nein!
    If M%=0 Or M%=2                  !  Nur im Lese-Modus
      *B%=Puffer$                    !    gelesene Daten aus dem lokalen
      '                              !    Puffer an den globalen übergeben
    Endif
  Endif
Return
```

Die nächste Prozedur ist innerhalb des Programms für die Überwachung der Tastatur zuständig. Hier wurde nur ein Teil

der Prozedur herausgeschnitten, da die anderen Teile zur Erläuterung des Disketten-Handlings unerheblich sind.

```
Procedure Control.
.
.
```

Für einen Disketten-Monitor ist es, wie oben schon erwähnt, lebenswichtig, jederzeit über die aktuelle Diskette informiert zu sein.

Aus diesem Grund wird hier durch 'BIOS(9)' festgestellt, ob in der Zwischenzeit die Diskette gewechselt wurde, um ggf. umgehend darauf reagieren zu können und sich die neuen Disketten-Grunddaten zu beschaffen.

```
  If Bios(9,0)                        ! Diskettenwechsel?
    If Mediach_flg%<>2                !  Media-Change-Control-Flag an?
      Al$="Programm-Unterbrechung !|Diskette wieder einlegen!"
      Al$=Al$+"|(Evtl. Schreibschutz öffnen)"
      Alert 3,Al$,1," OKAY ",B.bk%
      Alert 1,"MEDIA_CHANGE_CONTROL :",1," ON | OFF ",Mediach_flg%
      @Disktyp(*Sec%,*Spt%,*Sid%,*Spf%,*Fat2%,*Dlen%,*Dat1)
      '                               ! evtl. neue Disk-Daten
      Dec Sektor%                     ! Sektorzähler -1
      N_key%=1                        ! Abbruch-Flag setzen
      If Mediach_flg%=2               ! Media-Change-Control-Flag aus?
        Al$="MEDIA_CHANGE_CONTROL wird ab|der nächsten Sektor-Anzeige|"
        Al$=Al$+"mit der <UNDO> - Taste  | wieder aktiviert !"
        Alert 1,Al$,1," OKAY ",B.bk2%
      Endif
    Endif
  Endif
.
.
.
Return
```

Die Prozedur 'Lupe' hat eigentlich gar nichts mit Disketten zu tun. Sie ist einfach eine kleine Spielerei, um dadurch eine Anwendungmöglichkeit von Trick-Adressen zu demonstrieren. Wie

ich Ihnen bereits unter 'Spezial-Adressen' gezeigt habe, gibt es einen 64 Byte großen Speicherbereich, in welchem ständig der Maushintergrund zwischengespeichert wird. Dieser Bereich wird hier genutzt, um den Bereich, auf dem die Maus sich gerade befindet, in etwas vergrößerter Form darzustellen.

```
Procedure Lupe
  Local B.uf$,B.in$,I%,J%
  L.flg%=L.flg% Xor 1                    ! Lupen-Flag umschalten
  If L.flg%=1                            ! Lupe an?
    Get 568,308,568+64,308+64,L.back$    !  Hintergrund sichern
    Deffill ,0,0
    Pbox 568,308,568+64,308+32
    For I%=0 To 15                       !  16 Zeilen
      B.uf$=String$(32,"0")              !  Binärstring-Puffer
      B.in$=Bin$(Lpeek(10232+I%*4))      !  Maus-Zeile L-peeken
      Mid$(B.uf$,33-Len(B.in$),Len(B.in$))=B.in$  ! rechtsbündig in
      '                                  !  den Binärpuffer setzen
      For J%=1 To 32                     !  32 Bits pro Zeile
        If Mid$(B.uf$,J%,1)="1"          !    Bit gesetzt?
          Box 568+J%*2,308+I%*2,568+J%*2+1,308+I%*2+1 ! dann plotten
        Endif
      Next J%
    Next I%
  Else                                   !  Lupenflag aus!
    Put 568,308,L.back$                  !    Hintergrund restaurieren
  Endif
Return
```

Hhmmm, tja! Mir wäre am liebsten, wenn es diese Prozedur nicht gäbe, da sie meinen verzweifelten Versuch widerspiegelt, das System mit 100%iger Sicherheit dazu zu bringen, bei Directory-, FAT- oder Bootsektor-Manipulation sich alle (!) zur Diskettenverwaltung notwendigen Daten zu beschaffen. Dieses Kunststück ist mir nur in begrenzten Maßen gelungen. Die Empfindlichkeit des Systems in Beziehung auf Diskettenwechsel ist derart groß, daß bei vielen Datenänderungen (also Schreibaktionen) auf Disk sofort die Diskette als gewechselt angesehen wird. In bezug auf die Aktualisierung des Directorys läßt diese Empfindlichkeit sehr zu wünschen übrig. Deswegen war dieser Purzelbaum notwendig.

Für ein 100%iges Funktionieren kann ich allerdings trotzdem nicht einstehen. Bei Berücksichtigung des Umstands, daß ich nur drei Wochen Zeit hatte, um mich in diese schwierige Materie einzuarbeiten und diesen Monitor zu entwickeln, sollten mir diese relativ kleinen Mängel verziehen werden.

Sie können übrigens in sehr vielen Fällen Mißstände in der Directory-Verwaltung beheben, indem Sie tatsächlich die Diskette kurzfristig aus dem Laufwerk herausziehen, wieder hineinstecken und eine Funktion aufrufen, die eine Fileselect-Box verwendet (z.B. im Monitor die Funktionstaste 'F5' drücken).

```
Procedure Force_dt
  Void Bios(7,0)                    ! neuen Bios-Parameterblock holen
  Dpoke 2482,65535                  ! System-Media_Change_Flag setzen
  Pause 20                          ! Kontroll-Interrupt abwarten
  Open "O",#97,"Aux:"
  Dir "\*.*" To "Aux:"              ! Dummy-'Dir'-Aufruf in die Wüste
  Close #97
  Lpoke Xbios(14,0)+20,0            ! Aux:-Puffer wieder löschen.
Return
```

Mit der nächsten Prozedur ist es möglich, die gesamte Diskette, ohne sie formatieren zu müssen, in den Urzustand zu versetzen. Zwar sind die Sektoren dann noch mit Daten belegt, aber da hiermit die gesamten Directory- und FAT- Sektoren einfach komplett gelöscht werden, sieht der ST diese Diskette als frisch formatiert an.

```
Procedure Clr_dir.fat
  Bb$=Mki$(&HF7FF)+Chr$(&HFF)+String$(Spf%*512-3,0)! FAT-Aufbau
  @Secmod(3,Fat2%-Spf%,Spf%,0,0,Bb$)                ! FAT1 schreiben
  @Secmod(3,Fat2%,Spf%,0,0,Bb$)                     ! FAT2 schreiben
  Bb$=String$(Dlen%*512,0)                          ! Null-Sektoren
  @Secmod(3,Fat2%+Spf%,D.len%,0,0,Bb$)              ! Directory schreiben
Return
```

In der Einführung zu diesem Kapitel finden Sie eine Grafik, die den Aufbau eines Directory-Eintrags zeigt. Da hier, ebenso wie im Bootsektor, alle Word-Daten im MS-DOS-Format vorliegen

und außerdem die Zeit und das Datum der Dateierstellung bitweise in jeweils einem Word verschlüsselt sind, ist hier einiges an Rechnerei notwendig, um sich daraus brauchbare Daten zusammenzubasteln.

Das Directory wird hier in 32-Byte-Schritten solange durchgegangen, bis die Endmarkierung des Directories (4 Nullbytes am Anfang eines Eintrags) bzw. bei einem Unterdirectory das Ende des ersten Directory-Cluster erreicht wurde. Bei Unterdirectories ist es die Regel, daß die einzelnen Directory-Cluster an verschiedenen Stellen auf der Diskette liegen, da diese sich wie alle anderen Dateien nach den noch freien Clustern zu richten haben und diese unregelmäßig verstreut auf der Diskette liegen können. Um das Programm nicht noch umfangreicher werden zu lassen, habe ich mich deshalb auf die Anzeige der Einträge aus dem ersten SubDir-Cluster beschränkt.

Alle gefundenen Einträge werden nun analysiert und die Daten ausgegeben.

```
Procedure F3
.
.
.
  If No.s%=0                                  ! kein Abbruch?
    @Secmod(2,Sektor%,Dlen%,0,*Aa$,"")!   Directory-Sektoren lesen
    Pbox 4,4,634,267                          !   Fenster sauber
    Print At(3,1)                             !   Curso positionieren
    B.count%=0                                !   Eintrags-Zähler = Null
    Repeat                                    !   Einträge lesen
      B$=Mid$(Aa$,B.count%+1,11)              !   Dateiname
      If Cvl(Left$(B$,4))<>0                  !   ersten vier Zeichen leer?
        C%=Asc(Mid$(Aa$,B.count%+12,1))                   ! Attribut
        C$=Space$(3-Len(Str$(C%)))+Str$(C%)+" "           !  formatieren
        '  Mid$(Aa$,b.count%+13,10)
        '                          = 10 reservierte Nullbytes (unwichtig)
        '                            Diese Bytes können für versteckte
        '                            Informationen genutzt werden.
        D%=Asc(Mid$(Aa$,B.count%+26,1))*256               ! Datum Highbyte
        Add D%,Asc(Mid$(Aa$,B.count%+25,1))               ! Datum Lowbyte
        D$=Right$("0"+Str$(D% And 31),2)+"."              !|
        D$=D$+Right$("0"+Str$(D%/32 And 15),2)+"."        !|· formatieren
        D$=D$+Str$(D% Div 512+1980)+" "                   !|
```

```
E%=Asc(Mid$(Aa$,B.count%+24,1))*256              ! Zeit Highbyte
Add E%,Asc(Mid$(Aa$,B.count%+23,1))              ! Zeit Lowbyte
E$=Right$("0"+Str$(E% Div 2048),2)+":"           !|
E$=E$+Right$("0"+Str$(E% Div 32 And 63),2)+":"   !|- formatieren
E$=E$+Right$("0"+Str$(E%+E% And 63),2)           !|
F%=Asc(Mid$(Aa$,B.count%+28,1))*256              ! Startcluster HI
F%=F%+Asc(Mid$(Aa$,B.count%+27,1))               ! Startcluster LO
F$=Space$(4-Len(Str$(F%)))+Str$(F%)              !  formatieren
G%=(F%-2)*2+18                                   ! Startsektor
G$=Space$(4-Len(Str$(G%)))+Str$(G%)              !  formatieren
H%=Asc(Mid$(Aa$,B.count%+32,1))*2^24             ! Länge 1.Byte
H%=H%+Asc(Mid$(Aa$,B.count%+31,1))*2^16          ! Länge 2.Byte
H%=H%+Asc(Mid$(Aa$,B.count%+30,1))*2^8           ! Länge 3.Byte
H%=H%+Asc(Mid$(Aa$,B.count%+29,1))               ! Länge 4.Byte
H$=Space$(7-Len(Str$(H%)))+Str$(H%)              !  formatieren
I$=""
If C%=0                                          !__
  I$=I$+" Standard     "                         ! |
Endif                                            ! |
If C% And 1                                      ! |
  I$=I$+" Read Only    "                         ! |
Endif                                            ! |
If C% And 2                                      ! |
  I$=I$+" Hidden File  "                         ! |
Endif                                            ! |
If C% And 4                                      ! |- Attribut
  I$=I$+" System-File  "                         ! |  Bezeichnung
Endif                                            ! |
If C% And 8                                      ! |
  I$=I$+" Disk-Label   "                         ! |
Endif                                            ! |
If C% And 16                                     ! |
  I$=I$+" Ordner       "                         ! |
Endif                                            ! |
If C% And 32                                     ! |
  I$=I$+" Hardd.-Archiv"                         ! |
Endif                                            !--
If Left$(B$)=Chr$(229)                           !Löschflag
  I$=I$+"(gelöscht)"                             !(Sigma) gesetzt?
Endif
Eintrag$=B$+C$+D$+E$+F$+G$+H$+I$                 !Ausgabestring
Print At(4,Crslin);Eintrag$                      ! ausgeben
Add B.count%,32                                  !Eintragszähler
'                                                ! + 32 Zeichen
```

```
      If B.count% Mod 1024=0                          !Seite voll?
        @Wait                                         ! warten
        Pbox 4,4,634,267                              !Fenster klar!
        Print At(3,1)                                 !Cursor hoch
      Endif
    Endif
  Until Left$(B$,4)=Mkl$(0) Or (D.bk%=2 And B.count%=>992)
  ' Die Schleife wird erst verlassen, wenn entweder im Haupt-
  ' Directory kein Eintrag mehr gefunden wird, oder wenn bei
  ' einem Unter-Directory der erste Cluster ausgelesen wurde.
  @Wait
  Alert 2,"1. Directory-Cluster zeigen ?",2,"OKAY|NEIN",Bk%
Endif
.
.
.
Return
```

Für alle Disketten-Freaks kommt nun eine sehr interessante Prozedur.

Hier wird der Bootsektor analysiert. Interessant wird der Bootsektor jedoch erst, wenn man die darin enthaltenen Grunddaten auch verändern kann. Wenn Sie im Programm die Funktionstaste <F4> drücken, werden diese wichtigen Daten angezeigt. Sie können nun mit der Maus jede Zeile davon anfahren und erreichen dann mit einem Mausklick, daß Sie neue Daten in die Zeile eingeben können. Nachdem Sie 'ZURÜCK' angeklickt haben, werden Sie dann gefragt, ob die geänderten Daten auch in den Bootsektor auf Diskette geschrieben werden sollen.

Durch Änderung der Daten (z.B. Anzahl der Sektoren, Anzahl der Sektoren pro Track etc.) können Sie so spielend versteckte Sektoren erzeugen, indem Sie z.B. mehr Tracks formatieren und und mit Daten belegen, als vom System gelesen werden können. Wenn Sie nun aus einem Programm heraus diese Daten lesen wollen, ändern Sie erneut die entsprechende Angabe im Bootsektor und können nun auf die Daten zugreifen.

Haben Sie 81 oder 82 Tracks pro Seite installiert, können Sie sicher sein, daß bei einem normalen Desktop-Diskcopy diese Tracks nicht mitkopiert werden. Disketten, die also nicht über

die versteckten Daten auf den letzen beiden Tracks verfügen, können also dann nicht die Original-Disketten sein (Kopierschutz !!).

Welche Grunddaten im Bootsektor zu finden sind und welches Format sie besitzen, ist unter 'XBIOS(18)' beschrieben.

```
Procedure F4
.
.
.
  Restore Bootxt                    ! Data-Zeiger setzen
  For I%=1 To 14                    ! 14 wichtige Daten
    Read O.str$,Ofs1%,Hb%,Ofs2%     ! String,Byte-Offsets und Faktor lesen
    Print At(16,I%*2+2);O.str$;     ! String ausgeben
    If Hb%=65536                  !  Format > Word?
      Print Peek(Bt%+Ofs1%-1)*65536+Peek(Bt%+Ofs1%)*256+Peek(Bt%+Ofs2%)
      '                           !  3 Byte Seriennummer aus Puffer holen
    Endif
    If Hb%=0                      !  Fillstring?
      For K%=0 To Ofs2%           !   Stringbytes lesen
        Print Chr$(Peek(Bt%+2+K%)); ! und schreiben
      Next K%                     !   nächstes Byte
    Endif
    If Hb%=1                      ! Format = Byte?
      Print Peek(Bt%+Ofs1%)       !  Byte lesen und schreiben
    Endif
    If Hb%=256                    ! Format = Word?
      Print Peek(Bt%+Ofs1%)*(Hb%)+Peek(Bt%+Ofs2%)
      '                           ! Low <> High tauschen (MS-DOS-Format)
    Endif
  Next I%
  ' .
  ' .
  ' .
  If S.dummy$<>""                                  ! Eingabe gemacht?
    Deftext ,4,,6
    Bch.flg%=1                                     ! Änderungs-Flag an
    If Idx%=1                                      ! Fillstring?
      S_d$=String$(6,0)                            ! 6 Zeichen
      Mid$(S_d$,1,Len(S.dummy$))=S.dummy$          ! String einsetzen
      Print At(66,2+Idx%*2);S.dummy$''''''         !  ausgeben
      Lpoke Bt%+2,Cvl(Left$(S_d$,4))               ! -|und in Puffer
      Dpoke Bt%+6,Cvi(Right$(S_d$,2))              ! -|schreiben
```

```
    Else                                          ! numerische Eingabe!
      Print At(66,2+Idx%*2);Val(S.dummy$)''''''! Wert ausgeben
      High%=Val(S.dummy$)/256                     ! -|auf MS-DOS-Format
      Low%=Val(S.dummy$) And &X11111111           ! -|umrechnen
      If Hb%=4096                                 ! Seriennummer?
        Dpoke (Bt%+Ofs1%-1),High%                 !  Byte 1+2-|
        '                                         !  (MS-DOS) |in Puffer
        Poke (Bt%+Ofs2%),Low%                     !  Byte 3  -|
      Else
        If Ofs2%>0                                ! Word-Format?
          Poke (Bt%+Ofs2%),Low%                   ! High-Byte|
          '                                       ! (MS-DOS) |in Puffer
          Poke (Bt%+Ofs1%),High%                  ! Low-Byte |
        Else                                      !Byte-Format!
          Poke (Bt%+Ofs1%),Low%                   ! in Puffer schreiben
        Endif
      Endif
    Endif
  Endif
  ' .
  ' .
  ' .
Return
```

Was jetzt folgt, ist nichts für schwache Nerven.

Hier handelt es sich um einen echten Disketten-Dschungel, nämlich die FAT. Da sich durch ständiges Erstellen neuer und das Löschen alter Dateien die Cluster-Belegung auf der Diskette wie Spaghetti ineinander verknäult, haben sich die TOS-Bauer hier etwas ganz Besonderes einfallen lassen.

Um nicht für jede neue Datei, die angelegt wird, neuen Speicherplatz zu verschwenden, wird in der FAT jeder Cluster, der nicht bzw. nicht mehr benötigt wird, mit einem 12 Bit-Wert von Null gekennzeichnet. Wird nun eine Datei angelegt, geht das System die gesamte FAT durch und sucht der Reihe nach nach unbelegten Clusterplätzen. Dies macht es solange, bis die Anzahl der gefundenen Cluster (1 Cluster = 1024 Byte) ausreicht, um die Datenmenge der neuen Datei unterzubringen. Die Nummer des ersten gefundenen freien Clusters wird dann im Directory in das 27. und 28. Byte des neuen Directory-Eintrags geschrieben.

Da wir aber konfuserweise nicht im ST-Format arbeiten, sondern im MS-DOS-Format, steht im 27. Byte das LowByte und im 28. Byte das HighByte dieser Zahl. Um die Nummer des ersten belegten Clusters dieser Datei zu erfahren, muß man also diesen Wert aus dem Directory-Eintrag der betreffenden Datei auslesen. Wie das gemacht wird, habe ich oben unter 'Procedure F3' schon gezeigt.

Die durch die Datei belegten Cluster ermittelt man nun, indem man mit dieser Cluster-Nummer nun in die FAT geht. Und zwar an den FAT-Platz, der für diesen betreffenden Cluster reserviert ist. Dort steht nun die Nummer des nächsten von dieser Datei belegten Daten-Clusters. Nun springt man in den FAT-Platz der nun wieder für diesen Cluster reserviert ist usw., bis man auf einen Platzeintrag mit dem Wert '&HFFF' trifft. Dieser Wert stellt die Endmarkierung einer Datei dar. D.h., daß der Cluster, in dessen FAT-Platz der Wert '&HFFF' steht, der letzte dieser bestimmten Datei ist.

Um jetzt feststellen zu können, welches Byte innerhalb dieses letzten Daten-Clusters das letzte Byte der Datei ist, wird die Anzahl der gelesenen Cluster mit der im Directory eingetragenen Dateilänge verglichen. Ist die Datei z.B. 2466 Byte lang, muß also das 418. Byte (2466-2*1024) das letzte Byte der Datei sein.

Damit nicht genug! Daß ein Cluster zwei Sektoren umfaßt, wissen Sie bereits. Wenn eine Diskette 720 Sektoren hat, denkt man sich, daß demnach also 360 Cluster vorhanden sein müßten! Weit gefehlt! Im Normalfall hat eine 720-Sektoren-Diskette nur 351 Datencluster. Das liegt daran, daß die ersten 18 Sektoren (normalerweise!) für Bootsektor, FATs und Haupt-Directory ausgenommen sind. Da die Clusterzählung mit Null (bei Sektor 1/Track 0 der Diskette) beginnt, hat der erste zur Datenspeicherung freie Cluster die Nummer 2. Sektor 19 und 20 bilden also den ersten Daten-Cluster.

Aus diesem Grund sind die ersten beiden FAT-Plätze mit '&HF7FFFF' belegt, was soviel heißt wie: "Cluster 0 reserviert und Cluster 1 belegt".

Jaja, so einfach ist das!

Um das Chaos noch perfekt zu machen, müssen auch die Einträge der FAT-Plätze verdreht gelesen werden (MS-DOS). Es ist schon kompliziert genug, aus jeweils drei Byte für zwei FAT-Plätze immer die Halbbytes zu selektieren. Daß man nun auch noch die drei einzelnen Nibbels (je 4 Bit) vertauschen muß, setzt dem Ganzen die Krone auf.

Aber wozu gibt es Leute, die sich die Arbeit machen und fertige Algorithmen zur FAT-Analyse erarbeiten?

Einen solchen Algorithmus finden Sie in den nächsten beiden Prozeduren.

Eine Volksweisheit sagt: "Ein Bild sagt mehr als tausend Worte!" Frei nach diesem Motto folgt nun eine Grafik, die das Geschriebene hoffentlich verständlich werden läßt.

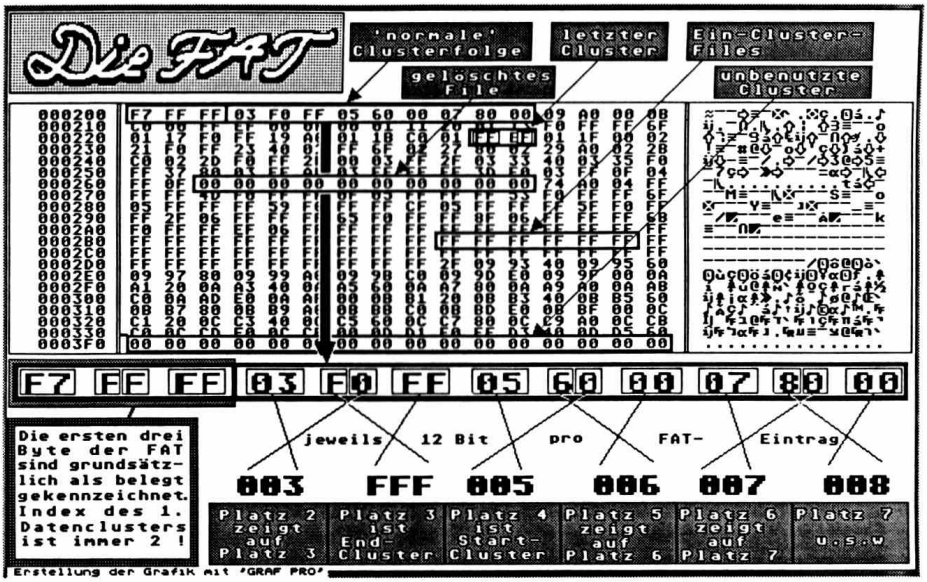


Abb. 7: Die FAT-Analyse

```
Procedure F_analyse(Cl%,Txt$)
  Local I%
  @Fat_array(*A.len%)              !Erstmal alle FAT-Einträge holen
  I%=Cl%-2                         !Startindex = Startcluster-2
  If I%<0                          !Startcluster im Directory = 0 ?
    Al$="Directory defekt ! Das File|wurde nach WRITE-Modus evtl."
    Al$=Al$+"|nicht korrekt geschlossen !"
    Alert 1,Al$,1," Aha ",F.bk%
    Sektor%=11                     !Schleifenzähler auf Directory-Start
  Else                             !gültiger Startcluster im Directory!
    Deftext ,,,4
    Deffill ,0,0
    Get 92,292,635,391,L.screen$   !Cluster-Anzeigefenster sichern
    Pbox 90,272,637,375
    Pbox 92,274,635,373
    If Fat%(I%)>0 And Fat%(I%)<&HFFF! Zeiger auf nächsten Cluster?
      Print At(17,47);"Weitere ";Txt$;"-Cluster: ";
    Else                           !Startcluster = Endcluster
      Print At(47,55);"Keine weiteren ";Txt$;"-Cluster !";
    Endif
    Cl.flg%=1                      !Cluster-Anzeige-Flag an
    Repeat
      If Fat%(I%)<>&HFFF And Fat%(I%)<>0 !nächster Cluster-Zeiger?
        If Crscol>103              ! Zeile voll?
          Print At(17,Crslin+1);   !  Zeilenumbruch am re. Fensterrand
        Endif
        Print Fat%(I%)'            ! Cluster ausgeben
      Endif
      J%=I%                        !FAT-Index merken
      I%=Fat%(I%)-2                !Sprung auf nächsten Cluster
    Until Fat%(J%)=&HFFF Or Fat%(J%)=0
    ' Schleife verlassen, wenn Endcluster erreicht oder kein neuer
    '                              !Zeiger gesetzt ist.
  Endif
  Deftext ,,,6
Return
```

Dies ist die Prozedur, die erstmal die gesamte FAT ausliest, in das richtige Zahlenformat 'verdreht' und die ermittelten Werte in ein eindimensionales Integer-Feld schreibt. Um sich besser orientieren zu können, beginnen die Platzeinträge mit Cluster 2 hier im Null-Element.

```
Procedure Fat_array(Arr_len%)
  Local I%,J%,Bb$,Low%,High%,Fat$,Byte1%,Byte2%
  Erase Fat%()
  Dim Fat%(Sec%/2)                          !Feldlänge=Clusteranzahl
  @Secmod(2,Fat2%-Spf%,Spf%,0,*Bb$,"")      !FAT-Sektoren lesen
  For I%=4 To (Sec%-18)/4*3 Step 3
```

Den Grund für den Aufbau dieses Schleifen-Limits habe ich oben in der Einführung zur 'Procedure F_analyse' schon angedeutet. Er ist abhängig vom 12-Bit-Format der FAT-Einträge und von der Differenz zwischen Cluster-Nummer und Feldindex (Index = Cluster-Nummer minus 2).

```
    Fat$=Mid$(Bb$,I%,3)                     !Einzelner FAT-Eintrag
    Byte1%=Asc(Left$(Fat$))                 !untere 2 Nibbel des
    '                                       !   vorderen Eintrags
    Low%=(Peek(Varptr(Fat$)+1)) And &X1111 !Low-Nibbel  |-des mittleren
    High%=Peek(Varptr(Fat$)+1)/16           !High-Nibbel |-   Bytes
    Byte2%=Asc(Right$(Fat$))                !obere 2 Nibbel des
    '                                       !  hinteren Eintrags
    Fat%(J%)=(Low%*256+Byte1%)              !12 Bit des vorderen Eintrags
    Fat%(J%+1)=(Byte2%*16+High%)            !12 Bit des hinteren Eintrags
    Add J%,2                                !Eintragszähler +2
  Next I%
  *Arr_len%=J%                              !echte FAT-Länge zurück
Return
```

Es folgt eine Prozedur, mit der Sie Ihre Disketten selbst formatieren können. Es ist oft lästig, den Interpreter verlassen zu müssen, um eine Diskette formatieren zu können. Mit etwas Geschick dürfte es Ihnen möglich sein, die für die Formatierung zuständigen Prozeduren aus diesem Programm herauszufiltern und auf eigene Bedürfnisse anzupassen.

Genauere Informationen zu diesem Thema finden Sie unter 'XBIOS(18)'.

```
Procedure F_dial
.
.
.
  If T.bk2%=1                          ! Disk komplett formatieren?
    T.bk%=1                            ! Formatierungs-Schleife auf!
    Ts%=0                              !  Starttrack =0
    Al$="Wieviel Sektoren pro Track?"
    Alert 2,Al$,1," 9 | 10 ",T.bk4%
    Al$="Einseitig oder zweiseitig?"
    Alert 2,Al$,1,"Single|Double",T.bk5%
    Sid%=T.bk5%
    But$=Str$(Sid%*80)+"|"+Str$(Sid%*81)+"|"+Str$(Sid%*82)
    Alert 2,"Wieviel Tracks?",1,But$,T.bk3%
    T%=80*Sid%+Sid%*(T.bk3%-1)-1
    ' 80 = minimale Trackanzahl pro Seite
    '                                |   Single    |    Double     |
    '                                |-------------|---------------|
    '    * Sid% (Seitenanzahl)   =>  |  80         |   160         |
    '    + Sid% mal  0, 1 oder 2 =>  |  0/1/2      |   0/2/4       |
    '                                |  80/81/82   |   160/162/164 |
    '    - 1        =>   (Zählerlimit)|  79/80/81   |   159,161,163 |
    '
    Spt%=8+T.bk4%                    ! Sektoren pro Track
    Alert 1,"Zu formatierende|Diskette einlegen!",1,"IS DRIN",T.bk.6%
  Endif
  If T.bk%=1                         ! Formatieren?
    For I%=Ts% To T%                 !  Starttrack bis Endtrack
      If Sid%=2                      !   Doppelseitig?
        Trk%=I% Mod (Sec%/Spt%/2)    !    gerade/ungerade Tracknummer
        If Odd(I%)                   !    ungerade?
          Sid_%=1                    !     dann auf Seite 2
        Else                         !    gerade!
          Sid_%=0                    !     dann auf Seite 1
        Endif
      Else                           !   Einseitig!
        Trk%=I%                      !    Tracknummer
        Sid_%=0                      !    auf Seite 1
      Endif
      Text 35,330,Str$(I%)+"  "
      @Formatter(Spt%,Sid_%,Trk%,*Nf%)! Formatieren
      If Nf%=2                       !  Formatierung abgebrochen?
        I%=T%                        !   Zähler auf Endtrack
        Flg%=1                       !   Abbruchflag setzen
```

```
      Endif
    Next I%                          ! nächster Track
    If T.bk2%=1 And Flg%=0           ! Komplett-Formatierung OK?
```

Aufbau des Bootsektors s. unter 'XBIOS(18)'.

```
      Boot$=Mki$(0)+String$(6,"N")
      Boot$=Boot$+Mki$(Random(65535))+Chr$(Random(255))
      Boot$=Boot$+Mki$(&H2)+Chr$(2)+Mki$(&H100)+Chr$(2)
      Boot$=Boot$+Mki$(&H7000)+Chr$(((T%+1)*Spt%) And &HFF)
      Boot$=Boot$+Chr$((T%+1)*Spt%/256)+Chr$(247+Sid%)
      Boot$=Boot$+Mki$(&H500)+Mki$(Spt%*256)+Mki$(Sid%*256)
      Boot$=Boot$+Mki$(0)+String$(30,"N")+String$(12,0)
      Boot$=Boot$+String$(3,245)+Chr$(&HFE)+Chr$(&H4F)
      Boot$=Boot$+Chr$(0)+Chr$(1)+Chr$(2)+Chr$(&HF7)+String$(22,"N")
      Boot$=Boot$+String$(12,0)+String$(3,245)+Chr$(&HFB)
      Boot$=Boot$+String$(391,0)+Mki$(&HABCD)
      '
      @Secmod(3,0,1,0,0,Boot$)       ! Bootsektor auf Disk schreiben
      @Clr_dir.fat                   !  FAT einrichten
      @Disktyp(*Sec%,*Spt%,*Sid%,*Spf%,*Fat2%,*Dlen%,*Dat1)
      '                              ! neue Disk-Daten holen
      @Force_dt                      ! DIR-Transfer erzwingen
      Dec Sektor%                    ! Schleifenzähler -1
    Endif
    Text 35,330,Str$(Int(Sektor%/Spt%))+"  "
    Flg%=0
  Endif
  If T.bk%=3 Or T.bk2%=3             ! Abbruch?
    Sh.flg%=1                        ! Rücksprungflag für 'Main' setzen
    Sput F.scr$                      ! Fullscreen restaurieren
  Endif
Return
```

Ein bißchen Komfort muß sein.

Haben Sie irrtümlich eine Datei gelöscht, können Sie sie wieder restaurieren, solange die entsprechenden Sektoren und FAT-Einträge nicht durch eine Folgedatei überschrieben wurden.

Allerdings sind hiermit nur Dateien auf der obersten Directory-Ebene restaurierbar. Dateien, die in einem Ordner lagen, können

hiermit nicht zurückgeholt werden, da die Suche nach den zusammengehörigen Daten-Cluster dadurch wesentlich erschwert würde. Grundsätzlich ist es jedoch auch möglich, Dateien aus SubDirs zu restaurieren. Wie man zu einer Datei bzw. zu einem Unterdirectory gehörende Cluster zusammensucht, wurde ja bereits in der Erläuterung zur 'Procedure F_analyse' angedeutet.

```
Procedure F8
  Erase B$()     !  -|
  Erase F%()     !  -| Felder löschen
  Erase H%()     !  -|
  Dim B$(112),F%(112),H%(112)
```

B$( ) Stringfeld für die Namen der gelöschten Dateien
F%( ) Integerfeld für Startcluster
H%( ) Integerfeld für Dateilängen

```
  Deffill ,0,0
  B.count%=0                              ! Bytezähler auf 0
  @Secmod(2,Fat2%+Spf%,Dlen%,0,*Bb$,"")! Haupt-Directory laden
  I%=0                                    ! Eintragszähler auf 0
  Repeat
    B$=Mid$(Bb$,B.count%+1,11)            ! Eintrag selektieren
    C%=Asc(Mid$(Bb$,B.count%+12,1))       ! File-Attribut
    If Cvl(Left$(B$,4))<>0 And Left$(B$)=Chr$(229) And C%<>8 And C%<>16
      '                    !gelöschter Eintrag/kein Label/kein Ordner?
      B$(I%)=B$                           ! Dateinamen eintragen
      F%(I%)=Asc(Mid$(Bb$,B.count%+28,1))*256       !Startcluster (High)
      F%(I%)=F%(I%)+Asc(Mid$(Bb$,B.count%+27,1))    !+Startcluster (Low)
      H%(I%)=Asc(Mid$(Bb$,B.count%+32,1))*2^24       !Länge   (Byte 4)
      H%(I%)=H%(I%)+Asc(Mid$(Bb$,B.count%+31,1))*2^16!+Länge  (Byte 3)
      H%(I%)=H%(I%)+Asc(Mid$(Bb$,B.count%+30,1))*2^8 !+Länge  (Byte 2)
      H%(I%)=H%(I%)+Asc(Mid$(Bb$,B.count%+29,1))     !+Länge  (Byte 1)
      Inc I%                              ! Eintragszähler +1
    Endif
    Add B.count%,32                       ! Bytezähler einen Eintrag weiter
  Until Left$(B$,4)=Mkl$(0)               ! Exit, wenn Directory-Ende
  If I%=0                                 ! keine gelöschte Datei gefunden?
    Pbox 108,346,544,364
    Print At(24,45);"Keine gelöschten Dateien vorhanden !"
    Pause 80
```

```
  Else                                  ! gelöschte Datei gefunden!
    ' .
    ' .
    ' .
    If X%>469 And X%<529                ! 'RECOVER'?
      Pbox 470,Y1%+1,529,Y2%            !-|
      Pause 10                          ! |- Blinker
      Pbox 470,Y1%+1,529,Y2%            !-|
      Deffill ,0,0
      Graphmode 1
      Pbox 108,346,544,364              ! Mini-Fenster klar
      Print At(27,45);"Restauriere File : ";"X";R.file$
      Sget F.screen$                    ! Screen sichern
      @Fat_array(*A.len%)               ! FAT auseinandernehmen
      @G_etclus(F%(J%),H%(J%),A.len%,J%) ! relevante Cluster feststellen
      Dec Sektor%                       ! Schleifenzähler -1
    Endif
  Endif
  ' .
  ' .
  ' .
Return
```

Während die letzte Prozedur die Aufgabe hat, alle Directory-Einträge zu selektieren, die mit der Lösch-Markierung '&HE5' beginnen und die FAT in ein Array zu übertragen, ist die nächste Prozedur dafür zuständig, die Anzahl der benötigten Cluster zu ermitteln, die FAT der Reihe nach nach leeren Plätzen zu durchsuchen und ggf. die richtige Clusterfolge der zu restaurierenden Datei wieder in die FAT einzutragen.

```
Procedure G_etclus(Cl_a%,Cl_e%,Arr_len%,P.os%)
  I%=Cl_a%-2                          ! Feldindex = Startcluster -2
  If I%<0 Or Cl_e%=0                  ! ungültiger Startcluster?
    '                                 !  oder Dateilänge = 0
    Al$="Directory defekt ?! Das File|wurde nach WRITE-Modus evtl."
    Al$=Al$+"|nicht korrekt geschlossen! |Restauration unmöglich !"
    Alert 1,Al$,1," Aha ",F.bk%
  Else                                ! Startcluster OK
    If Cl_e% Mod 1024=0               ! Dateiende genau auf Sektorende?
      Dec Cl_e%                       !   um 1 vermindern
    Endif
    Cl_e%=Int(Cl_e%/1024)+1           ! Anzahl der benötigten Cluster
```

```
If Fat%(I%)<>0                          !  Cluster schon belegt?
  Al$="File wurde bereits durch|Folgedatei überschrieben!"
  Al$=Al$+"|Restauration nicht mehr|möglich!"
  Alert 1,Al$,1," SCHADE ",Rs.bk%
  Goto No.restore
Else                                    ! Cluster ist frei!
  Deffill ,0,0
  Get 92,292,635,391,L.screen$          ! Cluster-Anzeigefenster sichern
  Pbox 90,272,637,375                   !  Fenster klar
  Pbox 92,274,635,373
  Cl.flg%=1                             ! Cluster-Anzeige-Flag an
  Mid$(Bb$,Instr(Bb$,B$(P.os%)))="X"+Right$(B$(P.os%),10)
  '                                     ! Dateinamen restaurieren
  @Secmod(3,Fat2%+Spf%,Dlen%,0,0,Bb$)
  '                                     ! Directory auf Disk zurück
  Box 587,355,635,373
  Print At(75,46);"MENÜ"
  Print At(15,45);"X";R.file$
  Deftext ,,,4
  Print At(17,47);"Belegte(r) Datei-Cluster: ";Cl_a%'
  If Cl_e%=1                            ! Nur ein Cluster nötig?
    Fat%(I%)=&HFFF                      !  Endmarkierung setzen
  Else                                  ! mehrere Cluster nötig!
    Mem%=I%                             !  letzten Cluster merken
    For L%=I%+1 To Arr_len%             !  FAT-Array durchgehen
      If Fat%(L%)=0                     !  nächste Cluster frei?
        Inc Ctrl%                       !    Clusterzähler +1
        If Crscol>103                   !    Cursor am Fensterrand
          Print At(17,Crslin+1);        !      dann neue Zeile
        Endif
        If Ctrl%<Cl_e%                  !    Zähler kleiner Clusteranzahl?
          Fat%(Mem%)=L%+2               !      Zeiger in Array eintragen
          Mem%=L%                       !      letzten Cluster merken
          Print L%+2'                   !      Cluster-Nummer ausgeben
        Else                            !    genug Cluster!
          Fat%(Mem%)=&HFFF              !     Ende in Array eintragen
          L%=Arr_len%                   !     Schleifenzähler auf Endlimit
        Endif
      Endif
    Next L%
  Endif
  S_fat$=Mki$(&HF7FF)+Chr$(&HFF)+String$(Spf%*512-3,0)
  '                                     !  FAT-Puffer vorbereiten
  For I%=0 To Arr_len% Step 2           !  FAT-Array durchgehen
    Byte1%=Fat%(I%) And &X11111111                    !-
    Byte2%=Fat%(I%)/256+((Fat%(I%+1) And &X1111)*16)!- MS-DOS-Format
```

```
        Byte3%=Fat%(I%+1)/16                                !-
        Dum.f$=Dum.f$+Chr$(Byte1%)+Chr$(Byte2%)+Chr$(Byte3%)
        '                                  !  FAT-Dummypuffer aufbauen
      Next I%                              ! Nächster Index
      Mid$(S_fat$,4,Len(Dum.f$))=Dum.f$ ! String in FAT-Puffer einsetzen
      @Secmod(3,Fat2%-Spf%,Spf%,0,0,S_fat$) !FAT 1 auf Disk schreiben
      @Secmod(3,Fat2%,Spf%,0,0,S_fat$)      !FAT 2 auf Disk schreiben
    Endif
    @Force_dt                            ! DIR-Transfer erzwingen
  Endif
  No.restore:
  Deftext ,,,6
Return
```

3. TOS-Allerlei

In diesem Kapitel finden Sie viele kurze Beispielprogramme, die sich wegen ihrer Kürze nicht auf der Diskette befinden.

Das Betriebssystem des Atari ST bietet eine fast unüberschaubare Fülle von Routinen und Funktionen an. In Entwicklungssystemen, die etwas auf sich halten, werden diese Routinen für den Programmierer verfügbar.

Es gibt sicherlich Literatur über diese Bibliotheken, es ist jedoch für den Ungeübten meist sehr schwer, wenn nicht sogar unmöglich, aufgrund dieser Literatur mit diesem Angebot an Routinen aus dem GFA-BASIC heraus etwas anfangen zu können.

Dies liegt hauptsächlich darin begründet, daß oft Grundkenntnisse in der Maschinensprache oder in 'C' vonnöten sind, um sich bei manchen Routinen ein Bild ihrer Arbeitsweise verschaffen zu können.

Um das Problem der Formatbestimmung und der Parameterreihenfolge etwas in den Griff zu bekommen, habe ich mir deshalb erlaubt, eine Möglichkeit des GFA-BASIC ausgiebig zu nutzen: Die Makros!

Bei Übergabe der Parameter an diese Kompakt-Funktionen brauchen Sie sich über das zu übergebende Zahlenformat keine Gedanken mehr zu machen.

Wenn man sich näher mit diesen selbstdefinierten Funktionen auseinandersetzt, kann man sich dadurch eine umfangreiche Bibliothek an echten Befehlserweiterungen schaffen. Die fertige Bibliothek speichert man mit 'Save,A' auf Diskette, lädt sie bei der Programmentwicklung ans Ende des Programmspeichers und kann nun diese Funktionen wie BASIC-Befehle verwenden. Was sie von echten Befehlen unterscheidet, ist der Umstand, daß ihr Aufruf entweder über 'VOID', als Zuweisung oder als Funktionsargument erfolgen muß.

Beispiel:

```
Void @Funktion(evtl.Parameter)
Print @Funktion(evtl.Parameter)
A%=@Funktion(evtl.Parameter)
A$=@Funktion$(evtl.Parameter)
A$=Str$(@Funktion(evtl.Parameter))
A%=Val(@Funktion$(Param.)+@Funktion$(Param.)
                          +Str$(@Funktion(Param.))
```

etc. Da der Mensch sich verständlicherweise eher an die Bedeutung von Namen als von Zahlen erinnern kann, ist es nach einiger Gewöhnungszeit im Umgang mit diesen Makros recht einfach, sich den Funktionsnamen und die evtl. zu übergebenden Parameter zu merken. Wenn man den 'Box'- oder 'Line'-Befehl verwendet, weiß man ja auch nach einiger Zeit, welche Parameter dieser Befehl erwartet, ohne jedesmal nachschauen zu müssen. Bei numerisch indizierten Funktionsnamen (z.B. XBIOS(15)..Bios(6)..Gemdos(22)) ist es schon wesentlich schwieriger, der Funktionsnummer eine bestimmten Bedeutung beizuordnen. Ich erinnere mich jedenfalls leichter an den Namen eines Freundes als an seine Telefonnummer. Es wäre auch schlimm, wenn es anders wäre.

Gefällt Ihnen der von mir gewählte Funktionsname nicht, können Sie ihn natürlich nach eigenem Gutdünken ändern.

Bei genauerem Hinschauen eröffnen sich hier ungeahnte Möglichkeiten, die das Programmieren zur reinen Freude werden lassen können.

Bei all den folgenden XBIOS-Funktionen habe ich dieses Prinzip konsequent eingehalten. Die Erklärung der Parameter bezieht sich immer nur auf das jeweilige Makro, nicht auf die dahinterstehenden System-Aufrufe, auch wenn die Parameter oft identisch sind. Eine Liste der verwendeten Funktionen finden Sie dann am Ende jedes Abschnitts. Diese Bibliotheken befinden sich auch auf der Diskette (XBIOSLIB.LST, BIOS_LIB.LST, GMDOSLIB.LST). Wie oben beschrieben, können Sie sie in Ihren

Arbeitsspeicher laden und sofort einsetzen. Ist Ihr Programm fertig entwickelt, löschen Sie alle Funktionen aus dieser Liste, die nicht zum Einsatz gekommen sind.

Es ist übrigens völlig (!!) egal, wo diese Funktionsdefinitionen (Deffn Name(Param.)) im Programm auftauchen, solange jede Funktion nur einmal definiert wurde. Die Bibliothek, bzw. ein einzelnes Makro kann also direkt am Programm-Anfang, in einer bestimmten Prozedur, am Programm-Ende oder an jeder beliebigen Programmstelle stehen.

3.1 XBIOS-Funktionen

Manche XBIOS-Funktionen (z.B. Settime, Gettime, Scrdmp) wurden hier nicht aufgeführt, da es dafür vollwertigen BASIC-Ersatz gibt (Settime, Date$, Time$, Hardcopy). Sie hier zu erwähnen wäre also überflüssig. Die XBIOS-Funktion 0 (Initmouse) wurde nicht aufgeführt, weil mir keine Situation bekannt ist, in welcher diese (eigentlich sehr wichtige) Funktion lauffähig eingesetzt wurde und ich sie, ehrlich gesagt, auch einfach nicht kapiere. Zu ihrem Einsatz sind äußerst umfangreiche und präzise Kenntnisse der Befehlsstruktur des Tastatur-Prozessors von Nöten, die ich nicht besitze. Die Definitionen finden Sie unter dem Namen "XBIOSLIB.LST" auf der Diskette.

Hier noch eine Liste der Fehlermeldungen, die das System bei nicht korrekt ausgeführten XBIOS-Funktionen zurückgibt.

XBIOS-Rückmeldungen

0	Kein Fehler aufgetreten
-1	Allgemeiner Fehler
-2	Station nicht empfangsbereit
-3	Unbekannter Befehl
-5	Ungültiger Befehl
-6	Track nicht gefunden
-7	Bootsektor nicht gültig
-8	Sektor nicht gefunden
-10	Schreibfehler

-12 Allgemeiner Fehler
-13 Floppy ist schreibgeschützt
-14 Floppy wurde gewechselt
-15 Gerät unbekannt
-16 Fehlerhafter Sektor
-17 Floppy nicht eingelegt

XBIOS(2)

Hiermit kann die Anfangsadresse des physikalischen Bildschirm-Speichers ermittelt werden. Der physikalische Screen ist der, auf welchem ein Bild angezeigt wird, also das Fenster in den Atari ST.

```
Pbase%=@Pbase
Deffn Pbase=XBIOS(2)
```

XBIOS(3)

Im Gegensatz zum physikalischen Screen existiert ein logischer Screen. Dieser ist im allgemeinen mit dem physikalischen Screen identisch. Solange man ihn nicht an eine andere Speicherposition verbannt (s. XBIOS 5). Alle Bildschirmausgaben werden auf den logischen Bildschirm bezogen. D.h., daß von den Ausgaben nichts zu sehen ist, wenn Physbase und Logbase mehr als 32 KByte auseinanderliegen.

```
Print @Lbase
Deffn Lbase=XBIOS(3)
```

XBIOS(4)

Hiermit wird die aktuelle Auflösung des Bildschirms ermittelt.

0 = 320 * 200 Punkte (LOWRES = 16 von 512 Farben)
1 = 640 * 200 Punkte (MIDRES = 4 von 512 Farben)
2 = 640 * 400 Punkte (HIRES = schwarzweiß)

```
Res%=@Reso
Deffn Reso=XBIOS(4)
```

XBIOS(5)

Bei BIOS(3) wurde schon angedeutet, daß die logische und physikalische Bildschirmadresse verschieden sein können. Dazu muß es natürlich eine Möglichkeit geben, diese Adressen zu bestimmen. Genau das passiert hier. Im ersten Parameter wird die logische Screen-Basis und im zweiten die physikalische Basis bestimmt. Der dritte Parameter der XBIOS-Funktion gilt der Bildschirmauflösung. Er kann hier vernachlässigt werden, da durch Änderung der Auflösung alleine keine sinnvolle Wirkung erzielt werden kann. Um diesen Parameter nutzen zu können, sind grundlegende Kenntnisse der Maschinensprache notwendig, um durch einen RESET das System mit der neuen Auflösung booten zu können. Die selbstdefinierte 'Setscr'-Funktion erwartet daher nur die ersten beiden Parameter. Die Bildschirmadressen müssen, um einen korrekten Bildaufbau zu erhalten, an einer durch 256 teilbaren Adresse liegen.

```
Deffn Setscr(L.ad%,P.ad%)=XBIOS(5,L:L.ad%,L:P.ad%,-1)
Deffn Get256(Adr%)=(Int(Adr%/256)+1)*256
```

Das neueste Fadenkreuz-Modell: >> FLEXI <<

```
On Break Cont
A$=Space$(32256)
Void @Setscr(@Get256(Varptr(A$)),@P.base)
A%=2
B%=3
Defline 6
Deffill ,2,4
Repeat
  Mouse X%,Y%,K%
  If X%<I%
    Sub I%,10
    Neu.x%=Abs(X%+10)
  Else
    Add I%,10
    Neu.x%=Abs(X%-10)
  Endif
  If Y%<J%
    Sub J%,10
    Neu.y%=Abs(Y%+10)
```

```
    Else
      Add J%,10
      Neu.y%=Abs(Y%-10)
    Endif
    I%=I% Mod 639
    Dpoke 9952,Neu.x%
    J%=J% Mod 399
    Dpoke 9954,Neu.y%
    Swap A%,B%
    Cls
    Line I%,0,X%,Y%-10
    Line X%,Y%+10,I%,399
    Line 0,J%,X%-10,Y%
    Line X%+10,Y%,639,J%
    Pcircle X%,Y%,7
    Circle X%,Y%,10
    Void @Setscr(XBIOS(A%),XBIOS(B%))
  Until Mousek Or Len(Inkey$)
  D%=Max(XBIOS(A%),XBIOS(B%))
  Void @Setscr(D%,D%)
```

Im Porsche durch den Speicher

```
  On Break Cont
  Deffill ,2,2
  Pbox 10,10,630,390
  Print At(34,12);"Taste drücken"
  Pbase%=@Pbase
  For I%=0 To Pbase% Step 256
    If I%>Pbase%-32000
      Sub I%,240
    Endif
    Void @Setscr(I%,I%)
  Next I%
  Void @Setscr(Pbase%,Pbase%)
  Void Inp(2)
  Edit
```

XBIOS(6)

Eine komfortable Möglichkeit, den Inhalt aller Farbregister gleichzeitig auszuwechseln. Dazu wird ein Puffer aus 16 Words vorbereitet, der nacheinander die Farbwerte für die einzelnen

Register enthält. Als Parameter wird der Funktion dann die Adresse dieses Puffers übergeben. Mit dem nächsten VBL-Interrupt wird diese Palette dann initialisiert.

```
A$=Mki$(2)+Mki$(333)+Mki$(521)+Mki$(1233)+Mki$(187)
A$=A$+Mki$(112)+Mki$(43)+Mki$(1561)+Mki$(233)+Mki$(78)
A$=A$+Mki$(1112)+Mki$(1403)+Mki$(161)+Mki$(1902)+Mki$(0)+Mki$(5)
Void @Setpal(Varptr(A$))
Edit
Deffn Setpal(Adr%)=XBIOS(6,L:Adr%)
```

XBIOS(7)

Diese Funktion wurde so aufbereitet, daß nur noch der Farbwert eines spezifizierten Farbregisters ausgelesen werden kann. Als Rückgabewert erhält man einen Wert ziwschen 0 und 1911 (&H0 - &H777). Als Parameter wird die Nummer des gewünschten Farbregisters übergeben.

Zur Bestimmung eines Farbregister-Inhaltes eignet sich der BASIC-Befehl 'Setcolor'.

```
C$=Hex$(@Getcol(1))
Print "Rot: ";Left$(C$),"Grün: ";Mid$(C$,2,1),"Blau: ";Right$(C$)
Edit
Deffn Getcol(Reg%)=XBIOS(7,Reg%,-1) And &HFFF
```

XBIOS(8)
XBIOS(9)

Es können ein oder mehrere Sektoren von/auf Diskette gelesen/geschrieben werden. Dabei ist ein Puffer mit ausreichender Größe einzurichten (512*Anz.d.Sektoren), der bei 'FLOPRD' die gelesenen Zeichen aufnimmt bzw. bei 'FLOPWR' die zu schreibenden Zeichen vor Funktionsaufruf enthält. Hier habe ich die beiden XBIOS-Routinen 8 und 9 zusammen mit der Routine 19 in einer gemeinsamen Funktion zusammengefaßt. Welche der drei Funktionen verwendet wird, wird nur durch die Übergabe der Funktionsnummer (8,9,19) bestimmt. Alle übrigen Parameter

haben bei allen drei Funktionen exakt dieselbe Bedeutung. Bei XBIOS(19) (Flopvf) muß der Puffer die Daten enthalten, die mit den Daten auf Diskette verglichen werden sollen.

Bei Aufruf sind folgende Parameter zu übergeben:

1. Funktionsnummer (Read=8/Write=9/Verify=19)
2. Anfangsadresse des Puffers
3. Nummer der anzusprechenden Floppy (0=A; 1=B)
4. Diskettenseite (SD=0 / DD=0 oder 1)
5. Nummer des Startsektors (Track-relativ 1-10, bzw. 1-11)
6. Nummer des Tracks (0 - 79/80/81)
7. Anzahl der Sektoren, die nacheinander gelesen/geschrieben/verglichen werden sollen.

Bei evtl. auftretenden Fehlern wird eine Fehlernummer von der Funktion zurückgeliefert (F%=@Flop..(...)). Bei 'FLOPVF' steht im Falle fehlerhafter Daten ab Pufferanfang eine mit einem Nullbyte abgeschlossene Word-Liste der fehlerhaften Sektoren.

```
A$=Space$(512*9)
Void @Floprwv(8,Varptr(A$),0,0,2,1,9)
' Hier müßte gegebenenfalls die Diskette gewechselt werden.
Void @Floprwv(9,Varptr(A$),1,0,32,5,4)
F%=@Floprwv(19,Varptr(A$),1,0,32,5,4)
If F%
  Print "Fehlerhafte Sektoren :"
  Repeat
    A%=Dpeek(Varptr(A$)+I%*2)
    Print A%
    Inc I%
  Until A%=0
Endif
Edit
Deffn Floprwv(M%,B%,D%,Si%,T%,S%,A%)=XBIOS(M%,L:B%,L:1,D%,S%,T%,Si%,A%)
```

XBIOS(10)

Wenn Sie dieses Kapitel gelesen haben, werden Sie in der Lage sein, Ihre Disketten selbst zu formatieren. Zuerst muß dazu natürlich die Diskette in Tracks eingeteilt werden. Das übernimmt diese Funktion. Sie können jeweils einen einzelnen Track for-

matieren. Wenn Sie das nacheinander mit allen Tracks gemacht haben, ist die Diskette formatiert. Nun muß noch der Bootsektor und die FAT installiert werden (s. XBIOS(18)), und Sie können die Diskette gewohntermaßen benutzen.

Dieser Funktion muß als erster Parameter die Adresse eines 8000 Byte großen Puffers übergeben werden, in dem die Track-Daten durch die Funktion erzeugt werden. Die selbstdefinierte 'Format'-Funktion erwartet noch 5 weitere Parameter:

2. Laufwerk (A: 0 / B:1)
3. Disk-Seite (einseitig 0 / zweiseitig 0 oder 1)
4. Nummer des zu formatierenden Tracks je Seite: 0-79, 0-80 oder 0-81
5. Anzahl der Sektoren pro Track (9 oder 10)
6. 'Virgin', ein Wordwert, mit welchem die Tracks beschrieben werden. Üblich: &HE5E5 oder 0. Kann jeder beliebige Wert unter 65535 sein (1027 ergibt ein nettes Pfeilchen-Muster).

Erwähnenswert ist noch, daß das System kein Fehlermeldung bei Überschreiten der maximalen Trackanzahl ausgibt. Also Vorsicht!

Wichtig: Vor Start dieses Programmes eine unformatierte oder unbeschriebene Diskette in Laufwerk A einlegen !!

```
A$=Space$(8000)
For I%=0 To 79
  F%=@Format(Varptr(A$),0,0,I%,9,&HE5E5)
  If F%
    Print "Fehlerhafte Sektoren in Track ";I%;" :"''
    Repeat
      A%=Dpeek(Varptr(A$)+I%*2)
      Print A%'
      Inc I%
    Until A%=0
  Endif
Next I%
Print "So sieht der Sektor 1 in Track 2 jetzt aus: ";Chr$(10)
Open "O",#1,"VID:"
A$=Space$(512)
```

```
Void Floprwv(8,Varptr(A$),0,0,2,1,1)
Print #1,A$
Close #1
Void Inp(2)
Edit
Deffn Format(B%,D%,Si%,T%,S%,V%)=XBIOS(10,L:B%,L:1,D%,S%,T%,Si%,1,
L:&H87654321,V%)
```

XBIOS(12)

Für alle, die die MIDI-Ports des ST nutzen wollen, ist das eine sehr wichtige Funktion. Der hier zu übergebende String wird als kompakte Einheit von der System-Funktion an den Midi-Out-Port übergeben. Noch einfacher geht die Daten-Übermittlung durch Midi allerdings mit dem 'Open "O",#1,"MID:" / Print #1,"xxx..", bzw. 'Open "I",#1,"MID:"' / Input #1,A$.

Übrigens ist es mit den beiden Midi-Buchsen problemlos möglich, zwei Computer durch zwei in HIFI-Geschäften überall erhältliche fünfpolige Diodenkabel direkt miteinander zu verbinden und kommunizieren zu lassen. Prinzipiell lassen sich sogar mehrere STs zu einem kompletten Netzwerk ausbauen, wobei allerdings die Midi-Thru-Buchse simuliert werden müßte. Dies geht wiederum nur, indem eine Maschine als 'Master-Keyboard' und die anderen mit gleichen Prioritäten als 'Slave-Keyboards' (Begriffe aus der Synthi-Szene) eingesetzt werden. Dazu sind den verschiedenen Geräten eigene Identifikationsziffern zuzuordnen, an welchen zu erkennen ist, an welche Maschine die aktuelle Information gerichtet ist. Ist sie nicht für die gerade empfangende Maschine gedacht, wird der empfangene Datenblock einfach an die nächste weitergegeben (MIDI-Thru). Mit ein bißchen Fantasie ist hier ein regelrechtes Super-Multitasking möglich. Bei mehr als 16 Maschinen müssen allerdings weitere 'Co-Master' eingesetzt werden, von denen jeder wieder 15 andere Maschinen steuern könnte. Aber wer hat schon 16 STs?

```
A$="Zu sendender Midi-String aus Bytewerten"
Void @W_midi(A$)
Deffn W_midi(Buf$)=XBIOS(12,Len(Buf$)-1,L:Varptr(Buf$))
```

XBIOS(14)

Ebenfalls für Midi-Nutzer und auch für RS232-Fans (DFÜ/ Mailbox) sehr interessant, ist die Möglichkeit, die Adressen und Betriebszustände der einzelnen Input/Output-Puffer zu ermitteln und auch zu verändern. Das folgende kleine Programm zeigt Ihnen, wo es dabei lang geht. Wollen Sie die Inhalte verändern, poken Sie die gewünschten Daten in den Vektor hinein.

Beispiel: RS232-Input-Puffer löschen

```
      Dpoke @Iobuff(0)+6,0
      Dpoke @Iobuff(0)+8,0

For J%=1 To 4
  Restore Puffer.datas
  For K%=1 To J%
    Read Zeile$,Of1%,Of2%
  Next K%
  Adresse%=@Iobuff(Of1%)+Of2%
  Print Zeile$'Lpeek(Adresse%)
  Restore Titel.datas
  For I%=4 To 12 Step 2
    Read Titel$
    Print Titel$," = "''Dpeek(Adresse%+I%)''"Byte"
  Next I%
Next J%
Deffn Iobuff(Dev%)=XBIOS(14,Dev%)
Puffer.datas:
Data "RS232 Eingabe-Puffer liegt ab Adresse: ",0,0
Data "RS232 Ausgabe-Puffer liegt ab Adresse: ",0,14
Data "Der Tastatur-Puffer  liegt ab Adresse: ",1,0
Data "Der MIDI - Puffer    liegt ab Adresse: ",2,0
Titel.datas:
Data Puffergröße,Head-Index bei,Tail-Index bei,Low-Water,High-Water
Edit
```

XBIOS(15)

Wieder etwas für DFÜ/Mailbox-Interessierte: die Möglichkeit, die RS232-Schnittstelle zu konfigurieren. Die selbstdefinierte Funktion erwartet 3 Parameter.

```
1. Baudrate (Baud%)  : 0=19200 / 1=9600 / 2=4800 / 3=3600
   (Bits pro Sek.)     4=2400  / 5=2000 / 6=1800 / 7=1200
                       8=600   / 9=300  / 10=200 / 11=150
                       12=134  / 13=110 / 14=75  / 15=50

2. Kontrollparameter : 0 = Handshake off (Default)
   (Ctrl%)             1 = XON / XOFF
                       2 = RTS / CTS
                       3 = XON/XOFF und RTS/CTS gleichzeitig

3. Usart-Register      Bit 1 und 2  = no parity     &X0....000
                                    = odd parity    &X0....100
                                    = even parity   &X0....110
                       Bit 3 und 4  = 1 Stopbit     &X0..01..0
                                    = 1.5 Stopbits  &X0..10..0
                                    = 2 Stopbits    &X0..11..0
                       Bit 5 und 6  = 8 Databits    &X000....0
                                    = 7 Databits    &X001....0
                                    = 6 Databits    &X010....0
                                    = 5 Databits    &X011....0
```

Wenn Sie einen der Parameter unverändert behalten wollen, übergeben Sie einfach eine -1.

```
Void Rs232(7,1,&X10111000)
Deffn Rs232(Baud%,Ctrl%,Usart%)=XBIOS(15,Baud%,Ctrl%,Usart%,-1,-1,-1)
```

XBIOS(16)

Diese Funktion bietet die äußerst reizvolle Möglichkeit, die Tastaturbelegung Ihres ST nach eigenen Bedürfnissen zu gestalten.

Es existieren nämlich drei Tabellen, die die ASCII-Codes der Tasten bei drei verschiedenen Zuständen beinhalten. Der erste Zustand ist, wenn 'normal' geschrieben wird, also weder <CapsLock> noch <Shift> gedrückt ist. Der zweite Zustand tritt ein, wenn eine <Shift>-Taste gedrückt wird und der dritte, wenn die <CapsLock>-Taste gedrückt wurde.

Die Tabellen sind je 128 Bytes lang. Ihre Adressen lassen sich nun durch diese Funktion ermitteln und ggf. auch bestimmen. Jede Tabelle ist 128 Bytes lang und enthält der Reihe nach die

ASCII-Codes der nach Scan-Code sortierten Tasten. Die Taste <F1> hat z.B. den Scan-Code 59. Wollen Sie, daß bei Druck auf diese Taste und gleichzeitig gedrückter Shift-Taste das Zeichen "A" ausgegeben wird, poken Sie in das Byte 59 der zweiten (Shift-) Tabelle den Wert 65.

Der folgende Fünfzeiler hilft Ihnen mit absoluter Sicherheit, den exakten Scan-Code jeder Taste zu erfahren.

```
Do
  Scancode%=@Scan
  Print Scancode%
Loop
Deffn Scan=Gemdos(8)/65536 And 255
```

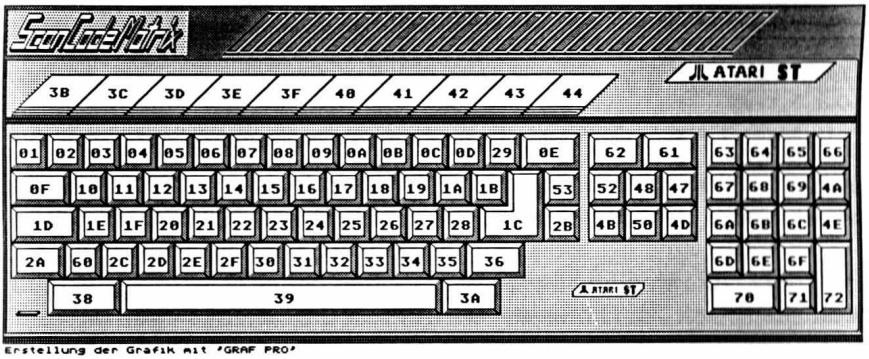


Abb. 8: Scan-Code-Matrix

Wollen Sie die Adresse des Zeigers auf die drei Tabellen ermitteln, bzw. soll eine der Tabellen unverändert bleiben, übergeben Sie der Funktion eine -1.

```
On Break Cont
Adr%=@Keyvec(-1,-1,-1)          ! Adr% enthält nun die Adresse der
'                               ! Unshift-Tabelle. Adr%+4 die der
'                               ! Shift- und Adr%+8 die der CapsLock-
'                               ! Tabelle
A$=Space$(128)                  ! 128 Byte-Puffer
Bmove Lpeek(Adr%),Varptr(A$),128! alte Tabelle in Puffer laden
```

```
B$=Chr$(242)+Chr$(243)+Chr$(244)+Chr$(245)+Chr$(246)+Chr$(247)
B$=B$+Chr$(251)+Chr$(252)+Chr$(253)+Chr$(254)
'                                ! 1. String (F1-F10) mit ASCII-Belegung
'                                ! bilden
C$=Chr$(23)+Chr$(24)+Chr$(25)+Chr$(20)+Chr$(21)+Chr$(22)
C$=C$+Chr$(17)+Chr$(18)+Chr$(19)+Chr$(16)
'                                ! 2. String (Ziffernblock-Zifferntasten)
'                                ! mit ASCII-Belegung bilden
'
For I%=17 To 25                  ! 3. String (Ziffernleiste-Zifferntaste)
  D$=D$+Chr$(I%)
Next I%
D$=D$+Chr$(16)
Mid$(A$,60,10)=B$                ! String 1 in Puffer einsetzen
Mid$(A$,104,10)=C$               ! String 2 in Puffer einsetzen
Mid$(A$,3,10)=D$                 ! String 3 in Puffer einsetzen
Void @Keyvec(Varptr(A$),-1,-1)   ! neuen Tabellen-Adresse übergeben
Print "F1 - F10 oder Tasten in Ziffernleiste/Ziffernblock drücken"
Repeat
  Key%=Inp(2)
  Out 5,Key%
Until Key%>25 And Key%<242
Void XBIOS(24)
Edit
Deffn Keyvec(U_shft%,Shft%,Caps%)=XBIOS(16,L:U_shft%,L:Shft%,L:Caps%)
```

Um bei riskanten Änderungsversuchen hinterher die Tastatur noch benutzen zu können, wird hier das Break unterbunden, damit auf jeden Fall XBIOS(24) ausgeführt wird. Dadurch wird der ursprüngliche Zustand wiederhergestellt, und die Tastatur kann wieder normal verwendet werden.

XBIOS(17)

Eine Zufallszahl wird durch die BASIC-Funktion 'Random(x)' auch geliefert. Diese XBIOS-Funktion habe ich nun so abgeändert, daß eine Bitzahl angegeben werden kann. Wird also als Bitzahl 4 übergeben, ist die höchste zurückgelieferte Zufallszahl eine 2^4 = 16. Bei 5 also 2^5 =32 usw. Die höchste verwendbare Bitzahl ist 23.

```
Do
  Print @Zufall(4)        ! 4 = 4 Bit-Zufallszahl (max.23 BIT)
Loop
Deffn Zufall(Lim%)=XBIOS(17) And ((2^(Lim% Mod 24))-1)
```

XBIOS(18)

Wie bereits bei XBIOS(10) und 'Procedure F_dial' des Disk-Monitors angekündigt, können Sie nach Lektüre dieses Kapitels Ihre Disketten selbst formatieren.

Das Formatieren selbst wird dabei durch XBIOS(10) übernommen. Damit eine Diskette verwendet werden kann, muß allerdings noch der Boot-Sektor und die FAT eingerichtet werden. Diese XBIOS-Funktion sollte eigentlich die Erstellung eines Boot-Sektors übernehmen. Vielleicht tut sie das ja auch, nur ist es mir nicht gelungen, mit Boot-Sektoren dieser Art eine Diskette zum Laufen zu bringen. Also machen wir uns den Boot-Sektor selber. Das ist ganz einfach, wenn man weiß, wie ein solcher aufgebaut ist. Dabei wollen wir uns hier nur um ein ganz normales Boot-Format kümmern. Sonderformate bleiben dann Ihrer Phantasie überlassen.

Bei allen Word-Werten ist zu beachten, daß sie im MS-DOS-Format angelegt sind, d.h. daß die High- und Low-Bytes miteinander zu vertauschen sind. Aus Mki$(512) = &H0200 wird dann ganz einfach Mki$(2) = &H0002.

```
Boot$=Mki$(0)+String$(6,"N")
```

Die ersten 2 Byte sind ein Zeiger auf das TOS-Boot-Programm ab Byte 31 des Boot-Sektors. Da ich nicht annehme, daß Sie Ihre TOS-Boot-Diskette selbst formatieren wollen, lassen wir diesen Zeiger unberücksichtigt, also Null.

Die Bytes 3-8 sind ein sogenannter 'Filler'. Der Inhalt dieser 6 Bytes ist unerheblich und kann zur versteckten Speicherung von Informationen genutzt werden. Hier habe ich sie auf 'N' gesetzt, wie es von der Desktop-Formatier-Routine gemacht wird.

```
Boot$=Boot$+Mki$(Random(65535))+Chr$(Random(255))
```

Die nächsten 3 Byte (24 Bit) enthalten eine 24-Bit-Zufallszahl, die als Seriennnummer interpretiert wird.

```
Boot$=Boot$+Mki$(&H2)+Chr$(2)+Mki$(&H100)+Chr$(2)
```

Byte 12 und 13 enthalten als Word die Anzahl der Bytes pro Sektor, üblicherweise den Wert 512 (&H2, s.o.).

Byte 14 als Byte die Anzahl der Sektoren pro Cluster, üblicherweise den Wert 2.

Byte 15 und 16 wieder als Word die Anzahl reservierter Sektoren Üblich: 1 (&H100 in MS-DOS s.o.). Damit ist der Boot-Sektor gemeint.

Byte 17 ist wieder ein Byte und enthält die Anzahl der FATs (File-Allocation Tables). Üblich: 2.

```
Boot$=Boot$+Mki$(&H7000)+Mki$(&HD002)
```

Byte 18 und 19 sind wieder ein Word, das die max. Anzahl der Directory- Einträge pro Directory (also auch Sub-DIRs) angibt. ' Üblich: 112 (&H0070 ST => &H7000 MS-DOS). Wenn Sie einen anderen Wert verwenden, sollten Sie darauf achten, daß er durch 16 teilbar ist.

Bytes 20 und 21 enthalten als Word die Gesamtanzahl der Sektoren auf der Diskette. Hier 720 (&H2D0 ST => &HD002 MS-DOS).

				Einseitig / Zweiseitig
80 Tracks je	9	Sektoren pro Seite	=	720 / 1440
80 Tracks je	10	Sektoren pro Seite	=	800 / 1600
81 Tracks je	9	Sektoren pro Seite	=	729 / 1458
81 Tracks je	10	Sektoren pro Seite	=	810 / 1620
82 Tracks je	9	Sektoren pro Seite	=	738 / 1476
82 Tracks je	10	Sektoren pro Seite	=	820 / 1640

```
Boot$=Boot$+Chr$(248)+Mki$(&H300)+Mki$(&HA00)+Mki$(&H100)
```

Das Byte 22 ist der sogenannte Media Descriptor. Er ist bei einseitigen Disketten auf 248 und bei zweiseitigen Disketten auf 249 festgelegt.

Byte 23 und 24 gibt als Word die Anzahl der Sektoren pro FAT an. Üblich: 5, hier: 3 (&H3 ST => &H300 MS-DOS).

Sie können also auch weniger als 5 FAT-Sektoren festlegen, solange für jeweils zwei Daten-Cluster der Diskette (1 Cluster=2 Sektoren) drei Byte Platz darin enthalten sind. D.h., bei 720 Sektoren = 360 Cluster geteilt durch 2 = 180 * 3 Byte = 540 Bytes. In diesem Fall müssen Sie also mindestens zwei Sektoren pro FAT einrichten.

Byte 25 und 26 ist wieder ein Word und enthält die Sektoren pro Track. Hier: 10 (&HA ST => &HA00 MS-DOS). Üblich sind 9 Sektoren pro Track.

Byte 27 und 28 ist noch ein Word mit der Anzahl der Seiten der Diskette. Einseitig: 1 (&H1 ST => &H100 MS-DOS), Zweiseitig: 2 (&H2 ST => &H200 MS-DOS).

```
Boot$=Boot$+Mki$(0)
```

Als letztes bedeutungsvolles Word wird in Byte 29 und 30 der Wert 0 eingetragen. Das bedeutet, daß es keine versteckten Sektoren gibt.

```
Boot$=Boot$+String$(30,"N")+String$(12,46)
Boot$=Boot$+String$(3,245)+Chr$(&HFE)+Chr$(&H4F)
Boot$=Boot$+Chr$(0)+Chr$(1)+Chr$(2)+Chr$(&HF7)
Boot$=Boot$+String$(22,"N")+String$(12,46)
Boot$=Boot$+String$(3,245)+Chr$(&HFB)+String$(391,46)
```

Dieser Block bedarf keiner Erklärung. Es handelt sich um die Bytes 31 bis 510, die keine erkennbare Bedeutung bei Normal-Disketten haben. Bei Disketten, mit denen das TOS gebootet werden soll, ist hier das dafür zuständige Boot-Programm zu finden.

Ich habe hier diesen Block so aufgebaut, wie es die Desktop-Formatier-Routine auch macht. Dabei ist im Aufbau kein Unterschied zwischen den verschiedenen Formaten festzustellen. Er ist immer gleich.

```
Boot$=Boot$+Mki$(&HABCD)
```

Zum Schluß wird in die Bytes 511 und 512 die Checksumme des Boot-Sektors eingetragen. Bei Normaldisketten ist der Inhalt dieses Words völlig unerheblich und kann zufällig gewählt werden.

Das Beispielprogramm zeigt Ihnen den oben selbstgeschneiderten Boot-Sektor an, verändert den Arbeitspuffer, indem diese XBIOS-Routine die in Zeile 4 gemachten Vorgaben darin einträgt, zeigt den neuen Boot-Sektor an, löscht anschließend den Puffer, läßt die XBIOS-Routine einen nagelneuen Boot-Sektor bauen und zeigt zum Schluß diesen Sektor an.

Daran ist zu erkennen, daß der Sektor vor Aufruf der Routine vorbereitet werden kann (muß?) und XBIOS daran außer den Vorgaben, die Sie beim Aufruf gemacht haben, der nicht (!) erzeugten Zufallszahl und der Checksumme keine Änderungen vorgenimmt. Die Zahlenkolonnen (0-9) dienen hier bei der Bildschirmausgabe nur als 'Lineal'.

```
Open "O",#1,"vid:"
Print String$(8,"1234567890");
Print #1,Boot$;
Void @Bootsek(Varptr(Boot$),-1,2)
Print Chr$(10);Chr$(13);String$(8,"1234567890");
Print #1,Boot$;
Void @Bootsek(Varptr(Boot$),-1,2)
Print Chr$(10);Chr$(13);String$(8,"1234567890");
Boot$=String$(512,0)
Void @Bootsek(Varptr(Boot$),2^25,2)
Print #1,Boot$;
Print Chr$(10);Chr$(13);String$(7,"1234567890");"123456789";
Void Inp(2)
Edit
Deffn Bootsek(Adr%,Ser%,Typ%)=XBIOS(18,L:Adr%,L:Ser%,Typ%,-1)
```

Die 'Bootsek'-Routine erwartet 3 Parameter:

- Adr% = Adresse des 512 Byte großen Arbeitspuffers

- Ser% = Eine 24-Bit-Seriennummer, die in die Bytes 9-11 eingetragen wird. Wird ein Wert > &HFFFFFF übergeben, wird eine zufällige Nummer von der Funktion erzeugt. Die Übergabe von -1 bewirkt, daß an der Seriennummer keine Änderungen vorgenommen werden.

- Typ% = Diskettentyp

  0 = MS-DOS-Format 40 Tracks SD
  1 = MS-DOS-Format 40 Tracks DD
  2 = ST-Format 80/81/82 Tracks SD
  3 = ST-Format 80/81/82 Tracks DD
  1 = Diskettentyp nicht verändern

Im Beispiel kann statt des selbsterzeugten Boot-Sektors auch einer von Diskette in den Puffer geladen werden (s. @FLOPRWV), mit XBIOS(18) nach Bedarf verändert und wieder auf Diskette zurückgeschrieben werden.

Wurde die Diskette erstmalig formatiert, muß der erzeugte Boot-Sektor auf der Diskette (immer Track 0, Sektor 0 auf Seite 0 !!) abgelegt werden. Dann muß allerdings noch die FAT eingerichtet werden.

Die FAT besteht bei einer neu formatierten Diskette aus lediglich den ersten drei Byte, die immer &HF7FFFF enthalten, und restlichen Nullbytes. Davon existieren in unserem Fall 2 Stück, die immer direkt hintereinander liegen und sich immer an den Boot-Sektor anschließen.

Im obigen Fall müssen also 2 mal 3 Sektoren (je 512 Byte = 3072 Byte) eingerichtet werden. Also:

```
A$=mki$(&HF7FF)+chr$(&HFF)+string$(1533,0)  ! 1.FAT
A$=A$+A$                                    ! 1.+ 2.FAT
Void @Floprwv(9,Varptr(a$),0,0,0,2,6)
```

Übrigens schließt sich direkt an die 2. FAT das Hauptdirectory an (s. Kapitel 'Diskettenmonitor').

Mit der Funktion '@Format' für alle Tracks und dem Aufbau des Boot- und der FAT-Sektoren ist Ihre Diskette nun gebrauchsfertig.

XBIOS(19)

Siehe unter XBIOS(8) / XBIOS(9)

XBIOS(21)

Ist es Ihnen auch schon mal passiert, daß Sie mit einer Textverarbeitung in einem großen Text herumgescrollt haben und hinterher nicht mehr wußten, wo der Cursor geblieben ist?

Hätte der Cursor geblinkt, würde man ihn sofort wiederfinden.

Mit dieser Funktion kann der TOS-Cursor vielfältig beeinflußt werden. Ich habe daraus 7 Einzelroutinen gemacht.

```
Deffn Curs_off=XBIOS(21,0)            !  Void @Curs_off = Cursor aus
Deffn Curs_on=XBIOS(21,1)             !  Void @Curs_on  = Cursor an
Deffn Blink_on=XBIOS(21,2)            !  Void @Blink_on = Cursor blinkt
Deffn Blink_off=XBIOS(21,3)           !  Void @Blink_off= Cursor steht
Deffn Sc_rate(Rate%)=XBIOS(21,4,Rate%)
'      Void @Sc_rate(Blinkrate)          setzt Cursor-Blinkfrequenz
Deffn Xsc_rate(Rate%)=XBIOS(21,4,Rate%)+@Curs_on+@Blink_on
'      Void @Xsc_rate(Blinkrate)         setzt Cursor-Blinkfrequenz,
'                                        schaltet Cursor und Blink-
'                                        Funktion an.
Deffn Gc_rate=XBIOS(21,5)             !  Rate%=@Gc_rate ergibt aktuelle
'                                        Blinkrate in 'Rate%'
' Beispiel:
Void @Xsc_rate(16)
Repeat
  A%=Gemdos(1) And 255    ! Gemdos(1) And 255 ergibt ASCII-Code der
  '                       ! gedrückten Taste in 'A%' und gleichzeitige
  '                       ! Ausgabe des entsprechenden Zeichens
```

```
  Print 'A%'              ! ASCII-Code ausgeben.
Until A%=13               ! Abbruch, wenn <Return>-Taste gedrückt.
Edit
```

XBIOS(24)

Siehe auch unter XBIOS(16)

Wurde mit XBIOS(16) (@Keyvec) die Tastaturbelegung geändert, kann hiermit der Urzustand wiederhergestellt werden, falls Sie nicht die Belegung so geändert haben, daß die normale Tastaturbelegung außer Kraft gesetzt ist und Sie keine 'vernünftigen' Zeichen mehr eingeben können. In kritischen Fällen sollten Sie bei Testläufen die Break-Funktion (Control/Shift/Alternate) unterdrücken (On Break Cont) oder gezielt zu einer Abbruch-Prozedur verzweigen (On Break Gosub...), welche dann vor Programmende XBIOS(24) ausführt. Es werden keine Parameter erwartet.

```
Deffn S_keys=XBIOS(24)
```

XBIOS(26)
XBIOS(27)

Ein MFP-Interrupt kann hiermit gesperrt (XBIOS(26) bzw. wieder zugelassen werden (XBIOS(27)).

Die @Mfp-Funktion erwartet 2 Parameter:

Parameter 1 = Modus (0 = sperren / 1 = zulassen)
Parameter 2 = Interrupt-Nummer (0 - 15 s. unten)

Vor Einsatz der ersten Zeile stellen Sie bitte Ihre Monitorlautstärke so, daß Sie die Tastaturklicks hören können.

```
Void @Mfp(0,5)
```

Der Tastatur-Repeat und die Klicks sind jetzt gesperrt. Sound-Verarbeitung ist ebenfalls gesperrt.

```
Sound 3,11,4,4
```

Control/Shift/Alternate drücken.

```
Void @Mfp(1,5)
Sound 3,11,4,4
```

Alles wieder okay.

```
Deffn Mfp(Mod%,Level%)=XBIOS(26+Mod%,Level%)
```

Der MFP-Chip fungiert im ST als Interrupt-Controller. In einem Vektor sind hier 16 Interrupt-Routinen aufgeführt, die nacheinander auszuführen sind.

0 = Centronics Strobe
1 = RS232 DCD
2 = RS232 CTS
(RS232-Empfänger bereit, Daten senden)
3 = nicht belegt
4 = RS232-Baudraten-Generator (Timer D)
5 = Timer C
(Systemtakt setzen, Sound-Verarbeitung)
6 = ACIA-Kontrolle
(Tastatur, MIDI, Joystick-Ports)
7 = Disk-Controller, DMA
8 = HBL-Zähler (Timer B)
9 = RS232 Output-Error
(Transmitter-Status in RS232 Parameterblock)
10 = RS232 Output-Buffer leer
(Sendebereitschaft, Sende-Register setzen)
11 = RS232 Input-Error
(Empfangs-Status löschen)
12 = RS232 Input-Buffer voll
(Data in Empfangpuffer schreiben)
13 = System Clock (Timer A)
14 = RS232 Ring-Indicator
15 = Monochrom Monitor Detect

XBIOS(28)

Es können mit dieser Funktion die Inhalte der Sound-Register ermittelt bzw. bestimmt werden. Bedeutung der Register s. XBIOS(32).

Die '@Giacc'-Funktion erwartet drei Parameter:

- Parameter 1 = Modus (0 = Reg. lesen / 1 = Reg. schreiben)
- Parameter 2 = Registernummer (0 - 15)

  Vorsicht: Register 14 und 15 haben nichts mit der Soundverarbeitung zu tun, sondern sind für die Floppy da (Funktion unbekannt, s. XBIOS(29)/XBIOS(30)). Man läßt sie also besser in Ruhe!
- Parameter 3 = Bytewert, der bei Modus 1 in das Register geschrieben werden soll.

```
Void @Giacc(1,0,15)
Void @Giacc(1,2,32)
Void @Giacc(1,4,57)
For I%=0 To 15
  Print "Inhalt des Soundregisters ";I%;" = ";@Giacc(0,I%,0)
Next I%
Deffn Giacc(Mod%,Reg%,Bte%)=XBIOS(28,Bte% Mod 256,128*Mod%+Reg% Mod 16)
```

XBIOS(29)
XBIOS(30)

Soundchip-Port-A-Bits setzen oder löschen.

```
Bit 0 = Disk-Seite 0 (@Ongibit(0)) oder 1 (@Offgibit(0)) wählen
Bit 1 = Diskstation A an (@On..(1)) / aus (@Off..(1))
Bit 2 = Diskstation B an (@On..(2)) / aus (@Off..(2))
Bit 3 = RS232 RTS (RequestToSend) an (@On..(3)) / aus (@Off..(3))
Bit 4 = RS232 DTR (DataTerminalReady) an (@On..(4)) / aus (@Off..(4))
Bit 5 = Centronics Strobe Busy (@On..(5)) / Stop (@Off..(5))
Bit 6 = GPO GeneralPurposeOutput-Pin an (@On..(6)) / aus (@Off..(6))
```

Die beiden selbstdefinierten Funktionen erwarten jeweils als Parameter die Nummer des Bits (0-6, s.o.), das gesetzt (@Ongibit(x)) oder gelöscht (@Offgibit(x)) werden soll.

```
For I%=1 To 20
  Void @Ongibit(1)
  Print At(10,10);"Diskette einlegen"
  Pause 5
  Void @Offgibit(1)
```

```
    Print At(10,10);"                    "
    Pause 5
  Next I%
  Edit
  Deffn Ongibit(Bit%)=XBIOS(29,Not 2^(Bit% Mod 7))
  Deffn Offgibit(Bit%)=XBIOS(30,2^(Bit% Mod 7))
```

XBIOS(32)

Mit dieser Funktion kann ein von Ihnen vorgegebener Sound-String auf Interrupt-Ebene abgearbeitet werden.

Dazu sind in den ersten 14 Registern des Soundchips bestimmte Werte zu setzen, die sich die Funktion aus dem String abholt. Außerdem können über drei Befehle, die in den String integriert werden, weitere drei Funktionen bestimmt werden, die von der benutzten Interrupt-Routine ausgeführt werden.

Die Struktur dieser XBIOS-Funktion ist nicht gerade einfach zu handhaben und liefert bei nur einem falsch gesetzten Register oder Befehl einen konfusen Klang- und Rauschsalat. Bei richtiger Handhabung können allerdings ganze Musikstücke mehrstimmig hiermit abgespielt werden.

In den String werden mit einem vorangestellten Identifikationsbyte versehene Datenbytes geschrieben, wobei die Reihenfolge gleichgültig ist, solange immer ein Bytepaar (eine Ausnahme!) zusammen in den String geschrieben wird.

Folgende Identifikationsbytes sind gültig:

0 = Das folgende Byte bestimmt mit acht Bit das LowByte der Periodendauer des 1. Soundkanals (0-255)

1 = Das folgende Byte bestimmt mit dem LowNibble (4Bit) das Highbyte der Periodendauer des 1. Kanals (0-15)

2 = Das folgende Byte bestimmt mit acht Bit das LowByte der Periodendauer des 2. Kanals (0-255)

3 = Das folgende Byte bestimmt mit dem LowNibble (4Bit) das Highbyte der Periodendauer des 2. Kanals (0-15)

4 = Das folgende Byte bestimmt mit acht Bit das LowByte der Periodendauer des 3. Kanals (0-255)

5 = Das folgende Byte bestimmt mit dem LowNibble (4Bit) das Highbyte der Periodendauer des 1. Kanals (0-15)

6 = Das folgende Byte bestimmt mit den unteren sechs Bit die Rauschgenerator-Frequenz (0-63)

7 = Das folgende Byte bestimmt, welche Soundkanäle aktiviert werden sollen. Dabei werden die Bitwerte für die aktiven Kanäle von dem Wert 255 abgezogen.

```
&X11111111 (255) = alle Kanäle aus
&X11111110 (254) = Kanal 1 an
&X11111101 (253) = Kanal 2 an
&X11111011 (251) = Kanal 3 an
&X11110111 (247) = Rauschen für Kanal 1 an
&X11101111 (239) = Rauschen für Kanal 2 an
&X11011111 (223) = Rauschen für Kanal 3 an
&X11000000 (192) = alle Kanäle mit Rauschen an
etc.
```

Achten Sie darauf, daß keine Werte kleiner als 192 verwendet werden, da sonst die Floppy-Ports im Soundchip angesprochen werden und dies unvorhersehbare Folgen haben kann.

8 = Das folgende Byte bestimmt in den unteren 4 Bit die Lautstärke für den Soundkanal 1 (0-15)

9 = Das folgende Byte bestimmt in den unteren 4 Bit die Lautstärke für den Soundkanal 2 (0-15)

10 = Das folgende Byte bestimmt in den unteren 4 Bit die Lautstärke für den Soundkanal 3 (0-15)

11 = Mit dem folgenden Byte wird die Feinabstimmung für die Länge der Hüllkurve (Sustain/Hüllkurvenfrequenz) vorgenommen (0-255).

12 = Mit dem folgenden Byte wird die Grobabstimmung für die Länge der Hüllkurve vorgenommen (0-255)

13 = Das folgende Byte bestimmt mit dem oberen Nibble die Art der Hüllkurve (8-15)

```
 8 = \|\|\|\|\  = Minus-Sägezahn, anfangs fallend
 9 = \________  = anfangs fallend, haltend
10 = \/\/\/\/\/ = Dreieckwelle, anfangs fallend
11 = \|________ = Attack/Cut, anfangs fallend
12 = /|/|/|/|/| = Plus-Sägezahn, anfangs steigend
13 = /________ = anfangs steigend, haltend
14 = /\/\/\/\/\ = Dreieckwelle, anfangs steigend
15 = /|________ = Attack/Cut, anfangs steigend
```

128 = speichert das folgende Byte (0-255) in einem Zwischenspeicher

129 = bildet die einzige Ausnahme bezüglich der Bytepaare. Diesem Identifikationsbyte müssen immer 4 weitere Bytes folgen. Diese Bytefolge bewirkt, daß der mit '128' zwischengespeicherte Wert in das mit dem ersten Folgebyte bestimmte Soundregister geladen wird und dieser Wert nach einer bestimmten Zeitspanne (4.Folgebyte / 50) im Register um den Wert des 2. Folgebytes erhöht wird. Im 3. Folgebyte wird ein Endwert bestimmt, bei dessen Erreichen dieser Prozeß abgebrochen wird. Byte 1 = betreffendes Soundregister (0-13) Byte 2 = Schrittweite (0-255) Byte 3 = Endwert (0-255) Byte 4 = Zeitintervall (0-255)

130 = Das folgende Byte gibt eine Zeitspanne an. Die Routine wird für die Zeit von 'Byte'/50 Sekunden verzögert bzw. bei der Übergabe einer Null beendet.

```
Deffn Dosound(Adr%)=XBIOS(32,L:Adr%)
```

XBIOS(33)

Es kann die aktuelle Druckereinstellung gelesen bzw. eine neue gesetzt werden.

Wenn man den Wert -1 übergibt, wird ein Bitvektor zurückgeliefert, der Auskunft über die aktuelle Einstellung gibt. Ein von -1 ungleicher Wert wird als neue Einstellung übernommen.

Dabei haben die ersten 6 Bits folgende Bedeutung:

	gesetzt	nicht gesetzt
Bit 1	Matrixdrucker	Typenraddrucker
Bit 2	Colordrucker	Schwarz/Weiß-Drucker
Bit 3	Atari	Epson
Bit 4	Test	Maximum
Bit 5	Centronics	RS232
Bit 6	Endlos	Einzelblatt

```
Deffn Printer(Attr%)=XBIOS(33,Attr%)
```

XBIOS(35)

Hiermit kann die Reaktionsverzögerung und der Repeat-Takt für den Tastaturklick eingestellt bzw. ermittelt werden. Damit ist das gemeint, was man auch mit den beiden Schiebereglern (Tastenfinger bzw. Hase und Schildkröte) im Control-Panel des

Desktops einstellen kann. Die Reaktionsverzögerung ist die Zeitspanne, die zwischen dem Druck auf eine Taste und dem Reagieren des Cursors vergeht. Mit dem Repeat-Takt ist die Spanne gemeint, die der Cursor nach der ersten Reaktion benötigt, um von einer Cursorposition zur anderen zu springen.

Im ersten Parameter wird entweder eine -1 (keine Änderung) oder die neue Verzögerungsrate erwartet, während der zweite Parameter -1 (keine Änderung) oder den neue Takt erwartet.

Die 'Normal'-Einstellung ist 'Re%' = 10 und 'Ta%'= 3.

Zurückgeliefert wird von der Funktion ein Longword, wovon die beiden Bytes des LowWords die Reaktionsverzögerung (HighByte) und den Takt (LowByte) darstellen. Der Wert 0 in 'Re%', bzw. 1 in 'Ta%' steht für schnell. Je größer der Wert, umso langsamer. Wird in 'Ta%' eine Null übergeben, ist der Tastatur-Repeat ausgeschaltet.

```
A%=@Takt(-1,-1) And &HFFFF
Re%=A% Div &HFF
Ta%=A% And &HFF
Deffn Takt(Re%,Ta%)=XBIOS(35,Re%,Ta%)
```

XBIOS(38)

Mit dieser Funktion kann eine Maschinenroutine, deren Adresse als Parameter zu übergeben ist, im Supervisor-Modus ausgeführt werden.

```
Deffn Supex(Adr%)=XBIOS(34,L:Adr%)
```

XBIOS(39)

Befindet sich das AES im RAM, kann es mit dieser Funktion abgeschaltet werden. Das Ergebnis ist ein Reset. Bei ROM-TOS-Maschinen konnte ich keine Wirkung feststellen.

```
Deffn Reset=XBIOS(39)
```

3.2 GEMDOS-Funktionen

Das GEMDOS ist die Hauptbibliothek zur Bereitstellung von grundlegenden Ein- und Ausgabefunktionen. Genau aus diesem Grund können wir hier auf einen Großteil der Funktionen verzichten, da sie wesentlich leichter indirekt über BASIC-Befehle verwendet werden können (z.B. Out, Input, Chdrive, Inp?, Inkey$ etc.).

All diese Routinen hier aufzuführen, würde nur Platz verschwenden.

Andere Routinen können innerhalb des GFA-BASIC unmöglich korrekt ausgeführt werden. Um den Leser nicht zu verwirren, wurde auch auf Erklärungen zu diesen Funktionen verzichtet. Unter dem Namen "GMDOSLIB.LST" finden Sie die Definitonen auf der Diskette.

Bei einigen der aufgeführten Routinen wird bei fehlerhaft ausgeführter Funktion ein Fehlercode zurückgegeben:

-32 ungültige Funktionsnummer
-33 angegebene Datei nicht gefunden
-34 angegebener Pfadname existiert nicht
-35 zuviele offene Dateien
-36 Datei-Zugriff ist nicht möglich
-37 ungültiges Datei-Handle
-39 Speicher ist nicht ausreichend
-40 ungültige Speicherblock-Adresse
-46 ungültiges Laufwerk
-49 weitere Dateien sind nicht vorhanden

Wird eine Null geliefert, heißt das, daß die Funktion korrekt ausgeführt wurde.

GEMDOS(1)
GEMDOS(7)

Diese beiden Funktionen sind eigentlich mit 'INP(2)' identisch. Der Unterschied ist zum einen der, daß hier ein 32-Bit-Wert

zurückgegeben wird. Ein weiterer Unterschied ist der, daß bei GEMDOS(1) gleichzeitig mit der Eingabe das eingegebene Zeichen an der aktuellen Cursorposition ausgegeben wird.

Das gelieferte Longword enthält in den unteren 8 Bit (A%=@Conin(1) And &HFF) den ASCII-Code und in den Bits 16-23 (A%=@Conin(1)/&H10000 And &HFF) den Scancode der Taste.

Da die beiden Funktionen eng verwandt sind, habe ich sie zusammen in einer 'Deffn'-Funktion untergebracht. Der zu übergebende Parameter bedeutet:

0 = nur Eingabe, keine Ausgabe
1 = Eingabe und gleichzeitige Ausgabe

```
Repeat
  A%=0
  For I%=1 To 9          ! Schleife wartet auf Eingabe von "GFA-BASIC"
    Add A%,@Conin(1)     !  offene Eingabe
    ' Add A%,@Conin(0)   !  verdeckte Eingabe
  Next I%
  Print
Until A%=289931869
Print Chr$(13);Chr$(10);A%
Edit
Deffn Conin(Mod%)=Gemdos(7-Mod%*7+Mod%)
```

Eine weitere Anwendungsmöglichkeit finden Sie unter 'XBIOS(16)'.

GEMDOS(25)

Diese Funktion ermittelt das aktuelle Laufwerk. Der zurückgelieferte '@Getdrv'-Wert bedeutet:

1 = Laufwerk A:
2 = Laufwerk B:

```
For I%=2 Downto 1
  Chdrive I%
  Print Chr$(13);"<RETURN> drücken";Spc(40);
  Fileselect "\*.*","",D$
  Print Chr$(13);"aktuelles Laufwerk = ";
  Print Chr$(64+@Getdrv);": (Taste drücken)";
  Void Inp(2)
Next I%
Edit
Deffn Getdrv=Gemdos(25)+1
```

GEMDOS(26)

Hiermit kann die 'Disk-Transfer-Adresse' bestimmt werden. Bei den GEMDOS-Funktionen 78 und 79 wird das Directory der aktuellen Diskette nach bestimmten Dateinamen durchsucht. Dazu wird ein Puffer benötigt, in dem die gefundenen Einträge eingetragen werden. Die Adresse dieses Puffers wird hiermit festgelegt. Er muß mindestens 42 Bytes umfassen. Mehr als 44 Byte sind allerdings nutzlos, da nur die ersten 42 Byte benutzt werden.

Das Beispiel zeigt eine Anwendung der drei genannten Funktionen.

```
Open "O",#1,"VID:"
A$=Space$(42)
Void @Setdta(Varptr(A$))
A%=@Sfirst("\*.*"+Chr$(0))
Repeat
  If A%
    Print "GEMDOS-Fehler: ";A%
  Else
    D$=Space$(12)
    Bmove @Getdta+30,Varptr(D$),12
    Size%=Lpeek(@Getdta+26)
    Print #1,A$;
    Print
    Print D$''"= ";Size%;" Bytes"
    A%=@Snext
  Endif
Until A%
Close #1
```

```
Deffn Sfirst(Adr$)=Gemdos(78,L:Varptr(Adr$),0)
Deffn Setdta(Adr%)=Gemdos(26,L:Adr%)
Deffn Getdta=Gemdos(47)
Deffn Snext=Gemdos(79)
Edit
```

GEMDOS(32)

Mit dieser Funktion hat man die Möglichkeit, die Supervisor-Kontrolle außer Kraft zu setzen, indem einfach im Supervisor-Modus gearbeitet wird. Es gibt zwei Möglichkeiten, diese Funktion ausführen zu lassen. Beim ersten Aufruf aus dem User-Modus kann der Wert Null übergeben werden. Als Rückgabewert erhält man den Stand des Supervisor-Stacks. Nachdem die Funktion mit dem Wert Null aufgerufen wurde, wird der Supervisor-Stack auf den Stand des User-Stacks gesetzt, und das Programm arbeitet nun im Supervisor-Modus.

Der nächste Aufruf sollte dann den Supervisor-Stack mit dem alten Stand restaurieren, um wieder ordnungsgemäß im User-Modus arbeiten zu können.

Wird als Wert beim ersten Aufruf aus dem User-Modus eine Adresse übergeben, nimmt das System diese Adresse als neue Superstack-Adresse an. D.h. der alte Supervisor-Stack ist 'eingefroren'. Bei diesem ersten Aufruf wird ebenfalls ein Longword zurückgegeben, und zwar die alte Adresse des Superstacks.

Beim nächsten Aufruf muß dann diese Adresse wieder an die Funktion übergeben werden, um wieder in den User-Modus zurück zu gelangen.

Beispiel 1

```
On Break Cont                    ! Abbruch unterdrücken
A%=@Super(0)                     ! alten Superstack holen
Bmove 0,XBIOS(2),32000           ! nur im Supervisor-Modus möglich!!
Print "SUPER-BMOVE aus Supervisor-Bereich = OKAY !"
Print "Supervisor-Stack ab Adresse: ",A%
B%=@Super(A%)                    ! alten Stand restaurieren
Pause 60
```

```
Cls
Print "USER-BMOVE aus Supervisor-Bereich = ERROR !"
Print "Supervisor-Stack (=USP) ab Adresse: ",B%
Bmove 0,XBIOS(2),32000          ! im User-Modus nicht möglich!!
Edit
```

Beispiel 2

```
On Break Cont                   ! Abbruch unterdrücken
A%=@Super(XBIOS(2)+920)         ! neue Supervisor-Stack-Adresse setzen
Print At(10,10);"SUPERVISORSTACK im Bildschirmspeicher !"
Label:
If I%<5000                      ! 5000 Errors
  On Error Gosub 123            ! zur Errorroutine
Else
  On Error Gosub Ende           ! der 5001. Error
Endif
Error I% Mod 100                ! Error 'ausführen'
Procedure 123
  Inc I%                        ! Zähler +1
  On Error                      ! Error-Gosub ausschalten
  Resume Label                  ! weitermachen
Return
Procedure Ende
  B%=@Super(A%)                 ! Superstack-Adresse restaurieren
  Print At(10,10);"SUPERVISORSTACK wieder zurück in Adresse ";A%
  Pause 100
  Edit
Return
Deffn Super(Mod%)=Gemdos(32,L:Mod%)
```

GEMDOS(47)

Mit der GEMDOS-Funktion 26 ist es möglich, die DTA zu bestimmen. Mit dieser Funktion dagegen kann die Adresse des aktuellen Disk-Transfer-Puffers ermittelt werden. Wie unter 'GEMDOS(26)' beschrieben, ist dieser Puffer 42 Byte groß.

Er hat folgenden Aufbau:

1. - 21. Byte	=	reserviert
22. Byte	=	File-Attribut
23. + 24. Byte	=	Uhrzeit der Datei-Erstellung (Word)

25. + 26. Byte = Datum der Datei-Erstellung (Word)
27. - 30. Byte = Dateigröße (erst Low- dann Highword)
31. - 42. Byte = Dateiname+Extension (incl.Trennpunkt)

```
A$=Space$(42)
Bmove @Getdta,Varptr(A$),42
Print A$
Deffn Getdta=Gemdos(47)
```

Ein weiteres Beispiel finden Sie unter 'GEMDOS(26)'.

GEMDOS(67)

Mit dieser Funktion läßt sich wieder hervorragend Versteck spielen. Sie können hiermit das Attribut einer Datei ermitteln oder auch verändern. Die Funktion erwartet drei Parameter:

Dat$ Name der Datei, deren Attribut ermittelt oder verändert werden soll.

Mod% gibt an, ob das Attribut ermittelt (0) oder geändert (1) werden soll.

Attr% gibt das Attribut an, das bei 'Mod% 1' dem File gegeben werden soll:

0 Normal (lesen/schreiben)
1 Nur-Lese-Datei
2 versteckte (hidden) Datei
4 System-Datei
8 Disketten-Label
16 Unter-Inhaltsverzeichnis

Beim Lese-Modus ist der 'Attr%'-Parameter unwichtig, sollte aber trotzdem angegeben werden. Grundätzlich wird bei jedem Funktionsaufruf als Rückgabe das aktuelle Attribut geliefert.

```
Print "BAS-Datei auswählen"
Fileselect "\*.BAS","",Dat$
O.atr%=@Change(Dat$,0,0)
```

```
If O.atr%=0
  Print "Altes Atrribut :";O.atr%
  Print @Change(Dat$,1,2)
  Pause 100
  Cls
  Print "Datei wird nicht mehr angezeigt"
  Print "Neues Atrribut :";@Change(Dat$,0,0)
  Fileselect "\*.BAS","",Dat2$
  Cls
  Print "Open ";Chr$(34);"I";Chr$(34);",#1,";Chr$(34);Dat$;Chr$(34)
  Open "I",#1,Dat$
  Print "Datei kann trotzdem geöffnet werden!"
  Print "Datei-Länge :";Lof(#1)
  Close #1
  Pause 200
  Cls
  Atrribut%=@Change(Dat$,1,0)
  Print "Attribut wieder auf 'Normal' gesetzt !"
  Print "Datei wird wieder angezeigt !"
  Fileselect "\*.BAS","",Dat2$
Else
  Cls
  Print "Gemdos-Fehler ";O.atr%
Endif
Edit
Deffn Change(Dat$,Mod%,Attr%)=Gemdos(67,L:Varptr(Dat$),Mod%,Attr%,0)
```

GEMDOS(72)

Diese Funktion ist mit dem BASIC-Befehl 'RESERVE' vergleichbar und wird auch von diesem benutzt. Es ist hiermit möglich:

1. die Größe des noch freien Benutzerspeichers zu ermitteln.

2. einen frei bestimmbaren Bereich ab dieser Adresse reservieren zu lassen.

Der so reservierte Bereich wird von nun an als belegt markiert und bei weiteren Speicherplatzvergaben (z.B. an andere Programme) ausgelassen.

Im BASIC ist von vornherein der gesamte Speicherplatz bis auf die letzten 16384 Bytes unterhalb des Bildschirmspeichers (XBIOS(2)) reserviert. Wird diese Funktion aufgerufen, ohne daß vorher mit RESERVE der BASIC-Speicher eingeschränkt wurde, werden genau diese 16384 Bytes als frei angzeigt.

Um mehr Speicher außerhalb des BASIC-Speichers zu reservieren, muß also vorher der RESERVE-Befehl verwendet werden. Als Parameter kann der Funktion eine -1 übergeben werden, wodurch die freie Speichergröße zurückgegeben wird. Wird ein Wert größer null übergeben, wird das als zu reservierende Speichergröße interpretiert und die Funktion gibt die Anfangsadresse des reservierten Bereichs zurück.

Hier habe ich eine Funktion in zwei 'Deffn'-Funktionen unterteilt. Die erste (@G_malloc) liefert die noch freie Größe. Hier braucht kein Parameter übergeben zu werden. Der zweiten (@S_malloc(Size%)) wird die gewünschte Speichergröße als Parameter übergeben. Sie liefert dann entweder die Startadresse des reservierten Bereichs oder eine Fehlermeldung (s.o.).

```
Rest%=@G_malloc
Print "Fre(0)=";Fre(0);" / Rest=";Rest%;" / Himem=";Himem
Reserve 200000
Print "Reservierung durch RESERVE !"
Rest%=@G_malloc
Print "Fre(0)=";Fre(0);" / Rest=";Rest%;" / Himem=";Himem
Print "Reservierung durch MALLOC !"
Size%=@G_malloc
Adr%=@S_malloc(Size%)
Print "freier Speicher ab Himem=";Size%;" startadr.=";Adr%
Rest%=@G_malloc
Print "Fre(0)=";Fre(0);" / Rest=";Rest%;" / Himem=";Himem
Print "MALLOC-Speicher wieder frei / BASIC-Speicher restauriert"
Void @Mfree(Adr%)
Reserve @Rfree
Rest%=@G_malloc
Print "Fre(0)=";Fre(0);" / Rest=";Rest%;" / Himem=";Himem
Edit
Deffn G_malloc=Gemdos(72,L:-1)
Deffn S_malloc(Size%)=Gemdos(72,L:Size%)
```

GEMDOS(73)

Wurde durch GEMDOS(72) oder GEMDOS(74) Speicher reserviert, kann er mit dieser Funktion wieder frei gemacht werden.

Dazu wird die Anfangsadresse des Bereichs übergeben. Diese Adresse muß mit der bei GEMDOS(74) bzw. GEMDOS(72) übergebenen Adresse übereinstimmen, damit keine Fehlermeldung zurückgegeben wird (ungültige Speicherblockadresse).

In diesem Zusammenhang finden Sie hier auch eine Funktion '@Rfree', die Ihnen die gesamte freie Speichergröße liefert, die für 'RESERVE' verfügbar ist. Dabei werden die 16384 Byte, die das AES unterhalb des Bildschirmspeichers benötigt, ausgenommen.

Den Einsatz dieser Funktion finden Sie in den Beispielen zu GEMDOS(72) und GEMDOS(74).

```
Deffn Rfree=XBIOS(2)-16384-Himem+Fre(0)
Deffn Mfree(Adr%)=Gemdos(73,L:Adr%)
```

GEMDOS(74)

Wenn bei BASIC-Start schon fast der gesamte Speicher reserviert ist und mit der vorherigen GEMDOS-Funktion nur Speicher hinter dem schon belegten Speicher reserviert werden kann, muß es auch eine Funktion geben, die diese Reservierung einschränken kann, um evtl. nicht mehr benötigten Speicher an das System zurückzugeben. Das wird durch diese Funktion bewerkstelligt, welche als Parameter zwei Werte erwartet:

Adr% beinhaltet die Anfangsadresse des Bereichs

Size% gibt an, wieviel Byte ab dieser Adresse reserviert werden sollen

Der darüberliegende Bereich wird an das System zurückgegeben und kann nun mit GEMDOS(72) für andere Aufgaben neu reserviert werden.

```
Rest%=@G_malloc
Print "Fre(0)=";Fre(0);" / Rest=";Rest%;" / Himem=";Himem
Reserve 200000
Print "200 KByte Reservierung durch RESERVE !"
Rest%=@G_malloc
Print "Fre(0)=";Fre(0);" / Rest=";Rest%;" / Himem=";Himem
Print "freier Speicher hinter HIMEM=";@G_malloc
A%=@S_malloc(20000)
Print "Startadresse des MALLOC-Bereichs=";A%
Print "freier Speicher hinter MALLOC-Bereich=";@G_malloc
Print "Setblock-Returnwert (0 = OKAY)=";@S_block(A%,16000)
B%=@S_malloc(-1)
Print "nächste freie Adresse hinter SETBLOCK=";B%
Void @Mfree(A%)
Reserve @Rfree
Print "BASIC-Speicher wieder restauriert! Himem=";Himem
Edit
Deffn S_block(Adr%,Size%)=Gemdos(74,0,L:Adr%,L:Size%)
```

GEMDOS(78)

Diese Funktion liefert Ihnen Information darüber, ob eine bestimmte, von Ihnen vorgegebene Datei im Directory der ebenfalls im Pfad anzugebenden Diskette enthalten ist oder nicht. Dazu wird ein String übergeben, der den Pfad- und Filenamen der gesuchten Datei enthält. Dieser String muß unbedingt (!) mit einem Nullbyte abgeschlossen sein.

Bei der Angabe des Dateinamens können sogenannte 'Wildcards' verwendet werden. D.h., daß für mehrere Zeichen, die nicht näher spezifiziert werden sollen, ein '*' und für einzelne Buchstaben, die bei der Suche unberücksichtigt bleiben sollen, ein '?' verwendet werden kann. Beispiel:

"\A????.LST" spricht auf der aktuellen Diskette alle Dateinamen an, die mit 'A' beginnen, fünf Zeichen lang sind und die Extension '.LST' tragen.

"B:\ORDNER\AB.*"* spricht auf Disk 'B:' alle Dateinamen an, die im Ordner '\Ordner' liegen, und mit den Buchstaben 'AB' beginnen. Hierbei ist die Länge des Namens und die Extension ohne Bedeutung.

Bei Verwendung dieser Wildcards wird von dieser Funktion der Name der ersten Datei geliefert, die der Suchbezeichnung entspricht.

Wurde die genannte Datei gefunden, werden im 'Disk-Transfer-Puffer' (s. GEMDOS(26) und GEMDOS(47)) die spezifischen Daten eingetragen. Wenn nicht, wird eine Fehlermeldung zurückgegeben. Ein Beispiel zu dieser Funktion finden Sie unter 'GEMDOS(26)'.

```
A%=@Sfirst("A:\GFA*.*"+Chr$(0))
Deffn Sfirst(Adr$)=Gemdos(78,L:Varptr(Adr$),0)
```

GEMDOS(79)

Diese Funktion sucht nach weiteren Dateien mit dem durch 'GEMDOS(78)' spezifizierten Namen. Es braucht kein Parameter übergeben zu werden. Wurden weitere Dateien gefunden, wird ebenfalls jedesmal der 'Disk-Transfer-Puffer' mit den dateispezifischen Daten gefüllt, wo sie dann ausgelesen werden können. Ein Beispiel hierzu finden Sie ebenfalls unter 'GEMDOS(26)'.

```
Deffn Snext=Gemdos(79)
```

3.3 BIOS-Funktionen

Die im weiteren Verlauf beschriebenen Definitionen finden Sie unter dem Namen "BIOS_LIB.LST" auf der Diskette.

BIOS(4)

Mit den XBIOS-Funktionen 8 und 9 haben Sie eine Möglichkeit kennengelernt, einzelne Sektoren von Diskette zu lesen bzw. auf Diskette zu schreiben.

Mit dieser Funktion haben Sie eine weitere Möglichkeit, das Gleiche zu tun. Der wesentliche Unterschied ist der, daß hiermit nicht trackrelativ, sondern diskettenrelativ gearbeitet wird. D.h., die angegebene Sektorennummer bezieht sich nicht auf den entsprechenden Sektor innerhalb eines Tracks, sondern auf einen Sektor innerhalb der gesamten Spanne an Sektoren, die auf der Diskette vorhanden sind.

Dazu wird ein Puffer eingerichtet, der die zu lesenden/schreibenden Daten erhält/enthält. Die Größe des Puffers muß 512mal so groß sein, wie Sektoren gelesen bzw. geschrieben werden sollen (1 Sektor=512 Byte).

Der Funktion sind fünf Parameter zu übergeben:

'Rwf%' bestimmt, ob Sektoren gelesen (0) oder geschrieben (1) werden sollen.

'Str%' enthält die Adresse des Puffers, aus dem die Daten gelesen bzw. in den sie geschrieben werden sollen.

'Anz%' enthält die Anzahl der Sektoren, die nacheinander gelesen/geschrieben werden sollen.

'Sec%' gibt an, mit welchem logischen Sektor begonnen werden soll (1 - max. Sektorenanzahl)

'Dev%' bestimmt das Laufwerk, auf welches sich die Funktion beziehen soll (0=A;1=B;2=Harddisk).

```
As%=5
A$=Space$(As%*512)
A%=@Rw_sek(2,Varptr(A$),As%,10,1)
Print A$
Deffn Rw_sek(Rwf%,Str%,Anz%,Sec%,Dev%)=Bios(4,Rwf%,L:Str%,Anz%,Sec%,
Dev%)
```

BIOS(7)

Hiermit kann die Adresse des BIOS-Parameterblocks ermittelt werden. Es handelt sich um einen 18 Byte großen Puffer, der für jedes Laufwerk getrennt angelegt wird. Darin enthalten sind grundlegende Daten zu dem jeweiligen Laufwerk bzw. zu der darin befindlichen Diskette.

Aufbau des 'BPB' (jeweils 1 Word):

Word 1	Bytegröße eines Sektors
Word 2	Anzahl der Cluster pro Sektor'
Word 3	Bytegröße eines Clusters
Word 4	Directory-Länge in Sektoren
Word 5	Größe eines FAT in Sektoren
Word 6	Sektornummer des 2. FAT
Word 7	Sektornummer des 1. Daten-Clusters
Word 8	Anzahl der vorhandenen Daten-Cluster
Word 9	verschiedene Flags (Bedeutung unbekannt)

Die Funktion erwartet als Parameter in 'Drv%' die Kennziffer des Laufwerks, dessen 'BPB'-Adresse geliefert werden soll (0=A: / 1=B:).

```
A%=@Get_bpb(0)
For I=0 To 7
  Print Dpeek(A%+I*2)
Next I
Edit
Deffn Get_bpb(Drv%)=Bios(7,Drv%)
```

BIOS(9)

Manchmal ist es interessant zu wissen, ob zwischenzeitlich die Diskette in einem der angeschlossenen Laufwerke gewechselt wurde oder nicht.

Dies können Sie durch diese Funktion erfahren. Dabei wird ihr ein Parameter ('Drv%') übergeben, der das Laufwerk angibt

(0=A: / 1=B:), das überprüft werden soll. Zurückgeliefert wird ein Wert:

0 = Diskette wurde nicht gewechselt
1 = Diskette möglicherweise gewechselt
2 = Diskette wurde gewechselt

```
A%=@Mediach(0)
Print A%
Edit
Deffn Mediach(Drv%)=Bios(9,Drv%)
```

BIOS(10)

Diese Funktion liefert Ihnen Information darüber, welche Laufwerke zur Zeit angeschlossen sind. Als Rückgabewert wird geliefert:

3 = A und B sind verfügbar
7 = A, B, und C sind verfügbar
11 = A, B, und D sind verfügbar
15 = A, B, C und D sind verfügbar

Das System geht grundsätzlich davon aus, daß mindestens die Laufwerke A: und B: angeschlossen sind. Auch wenn nur Laufwerk A: vorhanden ist, wird trotzdem der Wert 3 geliefert.

```
Print @Drvmap
Edit
Deffn Drvmap=Bios(10)
```

BIOS(11)

Diese Funktion bietet dem Programmierer die hervorragende Möglichkeit, seine Programmführung vom Status der Wechseltasten abhängig zu machen.

Damit sind die Tasten <Control>, <Shift li.>, <Shift re.>, <Alternate> und <CapsLock> gemeint. Hiermit kann der Status nicht nur ermittelt, sondern auch bestimmt werden.

Wird als Parameter 'Key%' eine -1 übergeben, wird von der Funktion als Rückgabe der aktuelle Status geliefert. Bei positiven Parameterwerten werden diese als neuer Status interpretiert.

1 = rechte Shift-Taste gedrückt
2 = linke Shift-Taste gedrückt
4 = Control-Taste gedrückt
8 = Alternate-Taste gedrückt
16 = CapsLock an

Es sind auch Tastenkombinationen möglich. Beispiel:

26 = <CapsLock> an + <Shift li.> + <Alternate>

```
Print "Keyboard-Shifter (Control,Shift,Alternate,CapsLock) drücken!"
Do
  Print At(10,10);@Kbshift(-1)'
Loop
Deffn Kbshift(Key%)=Bios(11,Key%)
```

3.4 VDISYS-Routinen

Die VDI-Bibliothek bietet einige sehr komfortable Routinen an, die es Ihnen erlauben, Ihre Programme vielfältiger auszustatten. Die Prozeduren sind im File "VDI_LIB.LST" zusammengefaßt.

Viele dieser Routinen sind vollwertig in BASIC-Befehle übernommen worden, so daß sie hier nicht aufgeführt werden brauchen.

Die folgenden Routinen bieten vor allem die Möglichkeit, verschiedene aktuelle Grafik-Einstellungen zu erfahren. Gerade für die Produktion von Utilities sind diese Routinen interessant, da Utilities die aktuellen Einstellungen ja nicht verändern dürfen bzw. diese nach Abschluß der Arbeiten wieder restaurieren müssen.

Als erstes stelle ich Ihnen eine echte Trick-Routine vor, die einen Mangel des VDI umgehen soll.

Es ist mir nämlich nicht gelungen, das für VDI-Aufrufe notwendige Handle zu ermitteln. Das Desktop erhält in fast allen Fällen das Handle 1. Welches Handle für Ihr jeweiliges Programm zuständig ist, hängt davon ab, wie viele Fenster von evtl. geladenen Accessories benutzt werden bzw. wie viele Fenster Sie im Programm verwenden.

Um nun mit Sicherheit feststellen zu können, mit welchem Handle die jeweilige Funktion aufzurufen ist, bleibt Ihnen nichts anderes übrig, als die folgende Prozedur aufzurufen.

Das Handle wird dadurch ermittelt, daß eine Schleife mit 16 Schritten (max. Handle-Nummer = 16) durchlaufen wird und in jedem Schritt eine 1 Pixel große Pbox gezeichnet wird. Vorher wird die VDI-Funktion 104 mit dem Schleifenindex als Handle ausgeführt. Diese Funktion hat den Zweck, die Umrahmung der Pbox auszuschalten. Logischerweise wird der Rahmen jedoch nur ausgeschaltet, wenn die richtige Handle-Nummer übergeben wurde. Ob der Rahmen bei dem gerade aktuellen Schleifenindex (=Handle) gelöscht wurde, wird mit dem 'POINT(X,Y)'-Befehl des BASIC überprüft. Das heißt also, daß der Schleifenindex als Handle übernommen werden muß, bei dem 'Point' eine Null liefert, also der Rahmen ausgeschaltet ist.

Da diese Prozedur die aktuelle Füllmuster-Einstellung sowie den Grafik-Modus verändert, sollte sie entweder am Anfang des Programms oder bei geöffneten Fenstern direkt nach Öffnung ausgeführt werden. Gegebenenfalls müssen die beiden genannten Grafik-Einstellungen hinterher restauriert werden, da die Prozedur sie noch nicht restaurieren kann. Sie verfügt ja noch nicht über das richtige Handle, sondern liefert es erst.

Damit das aktuelle Bild nicht gestört wird, wird vor Ausführung der 'Pbox'en der Screen-Teil, in dem sie erscheinen, zwischengespeichert und hinterher wieder restauriert.

Mit dem nun ermittelten Handle, das Ihnen in der Variablen 'H%' zurückgegeben wird, können Sie nun die VDI-Routinen ausführen lassen.

GETHANDLE

```
Procedure Gethandle(H%)
  Local Buff$
  Get 0,0,20,1,A$                 ! Screenteil zwischenspeichern
  Graphmode 1
  Local I%,Buff$
  Buff$=Mkl$(0)+Mkl$(1)+Mkl$(0) ! Contrl-Feld bilden
  Bmove Varptr(Buff$),Contrl,12 ! Contrl-Feld eintragen
  Dpoke Intin,1                   ! Rahmen an
  For I%=0 To 16                  ! erstmal Rahmen für
    Dpoke Contrl+12,I%            !   alle Handles
    Vdisys 104                    !   anschalten.
  Next I%
  Dpoke Intin,0                   ! Rahmen aus
  Deffill 1,0,0                   ! weiße Pbox
  For I%=0 To 16                  ! Pbox/Point-Schleife
    Dpoke Contrl+12,I%            !   Handle eintragen
    Vdisys 104                    !   Rahmen ausschalten
    Pbox I%,0,I%,0                !   Pbox zeichnen
    Exit If Point(I%,0)=0         !   Abbruch, wenn Rahmen nicht gesetzt
  Next I%
  Put 0,0,A$                      ! Screenteil restaurieren
  Dpoke Contrl+12,I%              ! richtiges Handle übergeben
  Dpoke Intin,1                   ! Rahmen an
  Vdisys 104                      ! Funktion ausführen
  *H%=I%                          ! Handle zurückgeben
Return
```

VDISYS 104 — Perimeter Visibility

Diese ist die Routine, mit welcher die Umrandungen bei Pbox, Prbox, Polyfill, Pcirle und Pellipse ausgeschaltet werden können.

Sie erwartet als Parameter das so wichtige Handle und ein Flag, das angibt, ob die Umrandung aus- oder eingeschaltet werden soll.

```
Procedure Visibel(Handle%,Flag%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(1)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
```

```
  Dpoke Intin,Flag%
  Vdisys 104
Return
```

VDISYS 114 **Fill Rectangle**

Manchmal ist es direkt ärgerlich, daß man in GFA-BASIC keine Möglichkeit hat, eine einfache Pbox ohne den manchmal lästigen Rahmen zeichnen zu können. Gerade für viele Arten der grafischen Aufbereitung von statistischen Daten (Diagramme) wäre dies von Wert.

Während Sie mit 'VDISYS 104' die Umrahmung aller 'P'-Objekte ausschalten, können Sie hiermit allein eine 'Pbox' ohne Umrahmung zeichnen.

Als Parameter werden wieder das Handle sowie die vier Koordinatenangaben der Box benötigt. Beispiel:

```
Deffill ,2,2
@Gethandle(*Bb%)
For I%=1 To 300 Step 2
  @Lbox(Bb%,I%,I%,I%+30,I%+30)
Next I%

Procedure Lbox(Handle%,Xl%,Yl%,Xr%,Yr%)
  Local Buff$
  Buff$=Mkl$(2)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Buff$=Mki$(Xl%)+Mki$(Yl%)+Mki$(Xr%)+Mki$(Yr%)
  Bmove Varptr(Buff$),Ptsin,8
  Vdisys 114
Return
```

VDISYS 129 **Clipping Rectangle**

Sie können mit dieser Routine einen bestimmten Bereich bestimmen, über den hinaus alle Grafikausgaben abgeschnitten werden. Sie benötigt dazu 6 Parameter:

1. das Handle
2. Modus (0=Clipping aus / 1=Clipping aktiv)

3.-6. X1 / Y1 / X2 / Y2

Bei den folgenden beiden VDI-Funktionen, die Box-Koordinaten erwarten, sind mit den ersten beiden Punkt-Parametern die linke obere Ecke und mit den nächsten beiden Punkt-Parametern die rechte untere Ecke dieser Box gemeint. Beispiel:

```
@Gethandle(*Bb%)
@Clipp(Bb%,1,240,150,400,250)
Circle 320,200,85
@Clipp(Bb%,0,240,150,400,250)
Circle 320,200,90

Procedure Clipp(Handle%,Flag%,Xl%,Yl%,Xr%,Yr%)
  Local Buff$
  Buff$=Mkl$(4)+Mkl$(1)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Buff$=Mki$(Xl%)+Mki$(Yl%)+Mki$(Xr%)+Mki$(Yr%)
  Bmove Varptr(Buff$),Ptsin,8
  Dpoke Intin,Flag%
  Vdisys 129
Return
```

Wie Ihnen vielleicht aufgefallen ist, übergebe ich das Control-Feld mit einem Bmove-Befehl. Das ist möglich, da die einzelnen Contrl-Elemente jeweils in Word-Abständen direkt hintereinander im Speicher liegen. Ich verpacke die einzelnen Contrl-Werte zu je zwei Werten mit 'Mkl$' in einem String und setze diesen String ganz einfach in das Contrl-Feld ein. Das hat eigentlich nur den Zweck, überflüssige Contrl-Pokes und damit Programmspeicher zu sparen.

Es folgen nun 4 Prozeduren, die die oben schon erwähnte Aufgabe haben, Ihnen Auskunft darüber zu geben, welche Grafikeinstellungen zur Zeit aktuell sind.

Die Bedienung dieser Prozeduren ist denkbar einfach. Bei Prozedur-Aufruf wird als erster Parameter das bereits genannte Handle übergeben, für das die Einstellungen erforscht werden sollen. Die folgenden Variablennamen, deren Anzahl von der

Prozedur abhängig ist, sind als Pointer zu übergeben. In diesen Variablen finden Sie nach Abschluß der Prozedur die Einstellungen.

Als da wären (jeweils der Reihe nach) :

Aus 'Procedure Getfill':

1. Füllfarbe
2. Fülltyp
3. Füllmuster
4. Grafikmodus
5. 'P'-Objektumrahmung (an/aus)

Aus 'Procedure Getmark':

1. Markerfarbe
2. Markertyp
3. Markergröße
4. Grafikmodus

Aus 'Procedure Getline':

1. Linienfarbe
2. Linientyp
3. Liniendicke
4. Grafikmodus

Aus 'Procedure Gettext':

1. Textfarbe
2. Textwinkel
3. Zeichenbreite
4. Zeichenhöhe
5. Breite der umfassenden Zeichenbox
6. Höhe der umfassenden Zeichenbox
7. Grafikmodus

Im Anschluß an die Routinen finden Sie ein Beispiel, daß die Übergabe und Auswertung demonstriert.

VDISYS 35 CURRENT POLYLINE ATTRIBUTS

```
Procedure Getline(Handle%,L1%,L2%,L3%,L4%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Vdisys 35
  *L2%=Dpeek(Intout)   !LINIENTYP
  *L1%=Dpeek(Intout+2) !LINIENFARBE
  *L4%=Dpeek(Intout+4) !GRAFIKMODUS
  *L3%=Dpeek(Ptsout)   !LINIENDICKE
Return
```

VDISYS 36 CURRENT POLYMARKER ATTRIBUTS

```
Procedure Getmark(Handle%,M1%,M2%,M3%,M4%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Vdisys 36
  *M1%=Dpeek(Intout)+1 !MARKERFARBE
  *M2%=Dpeek(Intout+2) !MARKERTYP
  *M4%=Dpeek(Intout+4) !GRAFIKMODUS
  *M3%=Dpeek(Ptsout+2) !MARKERGRÖßE
Return
```

VDISYS 37 CURRENT FILL AREA ATTRIBUTS

```
Procedure Getfill(Handle%,F1%,F2%,F3%,F4%,F5%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Vdisys 37
  *F2%=Dpeek(Intout)   !FÜLLTYP
  *F1%=Dpeek(Intout+2) !FÜLLFARBE
  *F3%=Dpeek(Intout+4) !FÜLLMUSTER
  *F4%=Dpeek(Intout+6) !GRAFIKMODUS
  *F5%=Dpeek(Intout+8) !PBOX-UMRAHMUNG
Return
```

VDISYS 38 CURRENT GRAPHIC TEXT ATTRIBUTS

```
Procedure Gettext(Handle%,T1%,T2%,T3%,T4%,T5%,T6%,T7%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(0)+Mkl$(0)+Mki$(Handle%)
  Bmove Varptr(Buff$),Contrl,14
  Vdisys 38
  *T1%=Dpeek(Intout+2)     !TEXTFARBE
  *T2%=Dpeek(Intout+4)     !ROTATIONSWINKEL
  *T7%=Dpeek(Intout+10)+1 !GRAFIKMODUS
  *T3%=Dpeek(Ptsout)       !ZEICHENBREITE
  *T4%=Dpeek(Ptsout+2)     !ZEICHENHÖHE
  *T5%=Dpeek(Ptsout+4)     !ZEICHENBOXBREITE
  *T6%=Dpeek(Ptsout+6)     !ZEICHENBOXHÖHE
Return
```

Beispiel:

```
@Gethandle(*Bb%)
Graphmode 2
Deffill ,2,4
Defmark 1,5,55
Deftext 0,16,900,12
Setcolor 0,1
Color 1
Defline 6,1,1,1
@Visibel(Bb%,0)
Pbox 10,100,400,300
@Getfill(Bb%,*Ff%,*Ft%,*Fm%,*Gm%,*Vs%)
Print "Füllfarbe:";Ff%;" Fülltyp:";Ft%;" Füllmuster:";Fm%
Print "Grafikmodus:";Gm%;" Pbox-Umrahmung:";Vs%
@Visibel(Bb%,1)
Box 200,200,600,380
@Getmark(Bb%,*Mf%,*Mt%,*Mh%,*Gm%)
Dim X%(1),Y%(1)
X%(0)=480
Y%(0)=40
Polymark 1,X%(),Y%()
Print "Markerfarbe:";Mf%;" Markertyp:";Mt%
Print "Markerhöhe:";Mh%;" Grafikmodus:";Gm%
@Gettext(Bb%,*Tf%,*Tr%,*Zb%,*Zh%,*Zbb%,*Zbh%,*Gm%)
Print "Textfarbe:";Tf%;" Textwinkel:";Tr%;" Zeichenbreite:";Zb%
Print "Zeichenhöhe:";Zh%;" Zeichenboxbreite:";Zbb%
Print "Zeichenboxhöhe:";Zbh%;" Grafikmodus:";Gm%
@Getcol(Bb%,0,*Cr%,*Cg%,*Cb%)
```

```
Print "Farbregister 0:";" Rot=";Cr%;" Grün=";Cg%;" Blau=";Cb%
@Getline(Bb%,*Lf%,*Lt%,*Ld%,*Gm%)
Print "Linienfarbe:";Lf%;" Linientyp:";Lt%;" Liniendicke:";Ld%
Print "Grafikmodus:";Gm%
Text 340,300,"'INQUIRE'"
Text 360,300,"(Nachfrage-"
Text 380,300,"Funktionen)"
Edit
```

Das folgende Beispielprogramm verwendet mehrere VDI-Routinen, die jede für sich im GFA-BASIC implementiert sind. Es handelt sich hier um Einstellungen der Linienart ('DEFLINE') oder der Textart ('DEFTEXT').

Warum ich sie Ihnen trotzdem vorstelle, hat den Grund, daß man mit diesen Funktionen auch die Linien- und Textarten des Desktop, der Alert- und Fileselect-Boxen, der RSC-Dialogboxen, der Pulldown-Menüs sowie auch teilweise der Fenster beeinflussen kann.

Ich habe diese Einstellungsfunktionen in zwei Prozeduren untergebracht. Sie sind beide von vornherein auf das System-Handle 1 eingestellt, also für andere Handle in dieser Form nicht verwendbar. Das ist aber auch nicht nötig, da innerhalb des BASIC die Grafik-Einstellungen wesentlich einfacher vorgenommen werden können.

Die erste Prozedur 'S_line' erwartet zwei Parameter. Der erste gibt an, welchen Linienstil Sie einstellen möchten. Mit den Werten 1-7 werden dieselben Linientypen festgelegt, wie Sie es vom BASIC her kennen. Mit dem Wert 0 wird die Linie unsichtbar gemacht und Werte, die kleiner als 0 sind, werden als 16-Bit-Muster für einen selbstdefinierbaren Linienstil verwendet.

Mit dem zweiten Parameter wird die Liniendicke festgelegt. Dabei gelten dieselben Werte wie bei 'DEFLINE'. Soll die Liniendicke unverändert bleiben, übergeben Sie einfach eine -1.

Der zweiten Prozedur 'S_text' werden drei Parameter übergeben. Damit wird wie bei 'DEFTEXT'

1. die Textart
2. der Textwinkel
3. die Texthöhe

angegeben. Soll eine dieser Einstellungen von der Prozedur nicht verändert werden, übergeben Sie ebenfalls -1.

Beispiel:

```
@S_line(0,-1)
Deffill ,2,4
Pbox 2,2,637,397
Deffill ,0,0
Pbox 9,100,420,280
Deftext ,20,,9
@S_text(5,-1,6)
Graphmode 2
Text 12,120,"Hier wurde einfach der Rahmen"
Text 12,140,"der Alert-Box"
Text 12,160,"ausgeschaltet."
Text 12,180,"Dasselbe läßt"
Text 12,200,"sich auch auf"
Text 12,220,"die Fileselect-"
Text 12,240,"Box anwenden."
Alert 1,"TEST        ",1,"OKAY|ABBRUCH",B%
Graphmode 1
Deffill ,2,4
Pbox 2,2,637,397
Deffill ,0,0
Pbox 100,10,540,380
Line 100,40,540,40
Deftext ,20,,26
Graphmode 2
Text 110,36,436,"D a t e i - A u s w a h l"
Fileselect "\*.*","",S$
```

VDISYS 15 / VDISYS 16 / VDISYS 113

```
Procedure S_line(Ls%,Ld%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(1)+Mkl$(0)+Mki$(1)
  Bmove Varptr(Buff$),Contrl,14
  If Ls%<7
    Dpoke Intin,Abs(Ls%)
    If Ls%<1
      Vdisys 113
    Else
      Vdisys 15
    Endif
  Endif
  Buff$=Mkl$(1)+Mkl$(0)
  Bmove Varptr(Buff$),Contrl,8
  If Ld%=>0
    Dpoke Intin,Ld%
    Vdisys 16
  Endif
Return
```

VDISYS 12 / VDISYS 13 / VDISYS 106

```
Procedure S_text(Ts%,Tw%,Th%)
  Local Buff$
  Buff$=Mkl$(0)+Mkl$(1)+Mkl$(0)+Mki$(1)
  Bmove Varptr(Buff$),Contrl,14
  If Ts%=>0
    Dpoke Intin,Ts%
    Vdisys 106
  Endif
  If Tw%=>0
    Dpoke Intin,Tw%
    Vdisys 13
  Endif
  If Th%=>0
    Dpoke Contrl+2,1
    Dpoke Contrl+6,0
    Dpoke Ptsin,0
    Dpoke Ptsin+2,Th%
    Vdisys 12
  Endif
Return
```

3.5 GEMSYS-Routinen

Im Kapitel 4 finden Sie schon eine umfangreiche AES-Bibliothek beschrieben. Diese Routinen beziehen sich fast ausschließlich auf die Objekt-, Menü-, Window- und Formular-Funktionen.

Das AES bietet darüber hinaus einige 'witzige' Funktionen an, die einem Programm erst den richtigen Schliff geben. Manche mag's stören, wenn beim Öffnen von Fenster auf dem Desktop kleine Rahmen über den Bildschirm huschen: "Überflüssiger Firlefanz!". Ich finde solche kleinen Bonbons ganz nett, da man um so mehr das Gefühl hat, es nicht mit einem seelenlosen Plastikkasten zu tun zu haben, auch wenn sie die Ablaufgeschwindigkeit etwas einschränken.

Deshalb hier nun einige solcher Routinen, die die Aufgabe haben, dem Bild etwas Leben einzuhauchen.

GEMSYS 70 **GRAF RUBBERBOX**

Die erste dieser Prozeduren produziert eine Punktlinien-Box, die an einer bestimmten Position mit gedrückter Maustaste initialisiert werden kann und deren rechte untere Ecke den Bewegungen des Mauszeigers folgt. So kann der Benutzer dazu gebracht werden, einen bestimmten Bildschirmbereich einzugrenzen.

Als Parameter erwartet die Routine zuerst die X/Y-Koordinaten der feststehenden, linken, oberen Boxecke. Die nächsten beiden Parameter geben in Pixel die Größe an, die die Box minimal einnehmen darf (erst Breite, dann Höhe).

In den beiden als Pointer gekennzeichneten Parametern erhalten Sie nach Loslassen der Maustaste die letzte Breite und Höhe der Box. Die Routine wartet hier selbständig auf den initialisierenden Mausklick, sie kann also aufgerufen werden, ohne daß die Maustaste gedrückt wurde.

Beispiel:

```
A%=10
B%=10
@G_rubber(A%,B%,10,10,*Br%,*Ho%)
Box A%,B%,A%+Br%,B%+Ho%
Procedure G_rubber(Xko%,Yko%,Xmin%,Ymin%,Lb%,Lh%)
  Repeat
  Until Mousek
  On Menu
  Local Buff$
  Buff$=Mki$(Xko%)+Mki$(Yko%)+Mki$(Xmin%)+Mki$(Ymin%)
  Bmove Varptr(Buff$),Gintin,8
  Gemsys 70
  *Lb%=Dpeek(Gintout+2)
  *Lh%=Dpeek(Gintout+4)
Return
```

GEMSYS 71 GRAF DRAGBOX

Diese Routine wartet ebenfalls auf einen Mausklick. Mit diesem Klick wird die Startposition eines kleinen Rechtecks bestimmt, das sich nur innerhalb eines zweiten, größeren Rechtecks bewegen läßt.

Als Parameter sind zu übergeben:

1. Breite der kleinen Box
2. Höhe der kleinen Box
3. X-Startposition der kleinen Box
4. Y-Startposition der kleinen Box
5. X-Koordinate der linken oberen Ecke der Grenzbox
6. Y-Koordinate der linken oberen Ecke der Grenzbox
7. Breite der Grenzbox
8. Höhe der Grenzbox

Nach Loslassen der Maustaste erhält man in den beiden 'Pointer-Variablen' die letzte X/Y-Koordinate der kleinen Box.

Beispiel:

```
A%=30
B%=30
C%=20
D%=20
Box C%,D%,C%+400,D%+150
@G_drag(A%,B%,Mousex,Mousey,C%,D%,400,150,*Lx%,*Ly%)
Box Lx%,Ly%,Lx%+A%,Ly%+B%
Procedure G_drag(Ltb%,Lth%,Ltx%,Lty%,Bgx%,Bgy%,Bgb%,Bgh%,Lsx%,Lsy%)
  Repeat
  Until Mousek
  On Menu
  Local Buff$
  Buff$=Mki$(Ltb%)+Mki$(Lth%)+Mki$(Ltx%)+Mki$(Lty%)
  Buff$=Buff$+Mki$(Bgx%)+Mki$(Bgy%)+Mki$(Bgb%)+Mki$(Bgh%)
  Bmove Varptr(Buff$),Gintin,16
  Gemsys 71
  *Lsx%=Dpeek(Gintout+2)
  *Lsy%=Dpeek(Gintout+4)
Return
```

GEMSYS 72 GRAF MOVEBOX

Soll der Effekt entstehen, daß sich ein Bildschirmobjekt von einer Position zu einer anderen bewegt, ist diese Routine dazu absolut prädestiniert. Sie übergeben als Parameter einfach die Breite und Höhe der Box, sowie die X/Y-Startposition und die X/Y-Zielposition (auf die linke obere Ecke bezogen).

Beispiel:

```
For I%=1 To 40
  X2%=Random(639-20)
  Y2%=Random(399-20)
  @G_move(20,20,X1%,Y1%,X2%,Y2%)
  X1%=X2%
  Y1%=Y2%
Next I%
Procedure G_move(Br%,Ho%,X1%,Y1%,X2%,Y2%)
  Local Buff$
```

```
  Buff$=Mki$(Br%)+Mki$(Ho%)+Mki$(X1%)+Mki$(Y1%)+Mki$(X2%)+Mki$(Y2%)
  Bmove Varptr(Buff$),Gintin,12
  Gemsys 72
Return
```

GEMSYS 73 **GRAF GROWBOX**
GEMSYS 74 **GRAF SHRINKBOX**

Diese Springboxen kennt wohl jeder. Sie erzeugen diesen dynamischen Effekt auf dem Desktop, wenn ein Fenster geöffnet oder geschlossen wird. Es wird entweder eine sich ausdehnende (73) oder zusammenschrumpfende Box (74) gezeichnet.

Von dieser Prozedur werden eine ganze Menge an Parametern benötigt.

1. Flag, ob die Box sich öffnen (1) oder schliessen soll (0)
2. u. 3. X/Y-Koordinate der jeweils kleineren Box
4. u. 5. Breite und Höhe der jeweils kleineren Box
6. u. 7. X/Y-Koordinate der jeweils größeren Box
8. u. 9. Breite und Höhe der jeweils größeren Box

Beispiel:

```
For I%=1 To 20
  A%=Random(140)
  B%=Random(100)
  X2%=Random(639)
  Y2%=Random(399)
  @G_groshr(1,X1%,Y1%,A%,B%,X2%,Y2%,639-X2%,399-Y2%)
  X1%=X2%
  Y1%=Y2%
  @G_groshr(0,X1%,Y1%,A%,B%,X2%,Y2%,639-X2%,399-Y2%)
Next I%
Procedure G_groshr(Flg%,Ltx%,Lty%,Ltb%,Lth%,Bgx%,Bgy%,Bgb%,Bgh%)
  Local Buff$
  Buff$=Mki$(Ltx%)+Mki$(Lty%)+Mki$(Ltb%)+Mki$(Lth%)
  Buff$=Buff$+Mki$(Bgx%)+Mki$(Bgy%)+Mki$(Bgb%)+Mki$(Bgh%)
  Bmove Varptr(Buff$),Gintin,16
  If Flg%=1
    Gemsys 73
```

```
  Else
    Gemsys 74
  Endif
Return
```

GEMSYS 52 FORM ALERT

Der 'ALERT'-Befehl des BASIC bietet diese GEMSYS-Routine an. Jedoch leider etwas eingeschränkt. Die BASIC-Alertbox kann maximal 4 Zeilen zu je 30 Zeichen und maximal 8 Zeichen je Klick-Button darstellen.

Die Original-Form-Alert-Box bietet dagegen 5 Zeilen zu je maximal 30 Zeichen und maximal 10 Zeichen je Button.

Dafür hat sie einen wesentlichen Nachteil! Sie stürzt einfach ab, wenn man in einer Zeile oder einem Button zuviele Zeichen eingibt. Und zwar so radikal, daß nur noch der Reset-Knopf aus der mißlichen Lage befreit.

In den allermeisten Fällen wird sich dieser Fehler jedoch vermeiden lassen. Die Syntax des Aufrufs ist genauso gehalten, wie Sie es vom 'ALERT'-Befehl her gewohnt sind.

Beispiel:

```
Al$=String$(30,"1")+"|"+String$(30,"2")+"|"+String$(30,"3")
Al$=Al$+"|"+String$(30,"4")+"|5 Zeilen zu je 30 Zeichen"
@F_alert(1,Al$,2," und je 10|Zeichen|Buttontext",*Button%)
Print Button%
Procedure F_alert(Symbol%,Albxtxt$,Knopf%,Albttxt$,Sl%)
  Local Al$
  Al$="["+Str$(Symbol%)+"]["+Albxtxt$+"]["+Albttxt$+"]"+Chr$(0)
  Lpoke Addrin,Varptr(Al$)
  Dpoke Gintin,Knopf%
  Gemsys 52
  *Sl%=Dpeek(Gintout)
Return
```

4. Objekte, Formulare und Resources

Leider ist bisher unter GFA-BASIC die GEM-Oberfläche nur bedingt ansprechbar. Befehle und Funktionen, die die Resource-Library, die Formularroutinen, die Object-Ansprachen usw. betreffen, fehlen oder sind nur unzureichend implementiert. Das soll jetzt anders werden.

Auch wenn es vielleicht zuerst verwirren wird, um etwas tiefere Einblicke in die Objektstrukturen kommt man nicht herum. Hierarchische Bäume, Speicherfelder, Word- und Longwordpointer werden Ihnen aber nach Lektüre dieses Kapitels leicht von den Lippen fließen.

4.1 Objektgenerierung in GFA-BASIC

Im Desktop tauchen nach Anklicken einiger Menüpunkte die berühmten kleinen Bildchen auf, die Sie über Betriebszustände unterrichten, vor etwas warnen oder Eingaben verlangen. Alle diese Bildchen sind GEM-Objektbäume, die wiederum aus GEM-Objekten bestehen. In GFA-BASIC kennen Sie die Alert-Box, die auch nichts anderes ist als ein Objektbaum, bestehend aus den Unter-Objects <ICON>, ein, zwei oder drei <STRINGs> und ein bis drei <BUTTONs>.

Damit GEM diese Objekte entsprechend verwalten kann, müssen sie nach ganz bestimmten Regeln aufgebaut werden.

Regel 1 Alle Objekte werden in regelmäßigen Strukturen, Feldern, speicherintern abgelegt. Eine Objektstruktur ist genau 10 Words und ein Longword, also (10*2)+(1*4) = 24 Bytes, groß. Im Speicher liegen die Objekte der Reihe nach hintereinander.

Regel 2 Die Objekte bauen in Form eines Objektbaums hierarchisch aufeinander auf. Dazu hat jede Objektstruktur in den ersten drei Words bestimmte Zeiger, von

denen der erste auf das nächste Objekt derselben Hierarchie zeigt und die anderen zwei auf das erste und letzte der nächsten Generation.

Regel 3 Neben der Objektstruktur sind verschiedene andere Strukturen vorhanden, die nach demselben Muster aufgebaut sind und Spezialinformationen über das entsprechende Objekt beinhalten. Das Longword der Objektstruktur trägt die Anfangsadresse der jeweiligen Sonderstruktur.

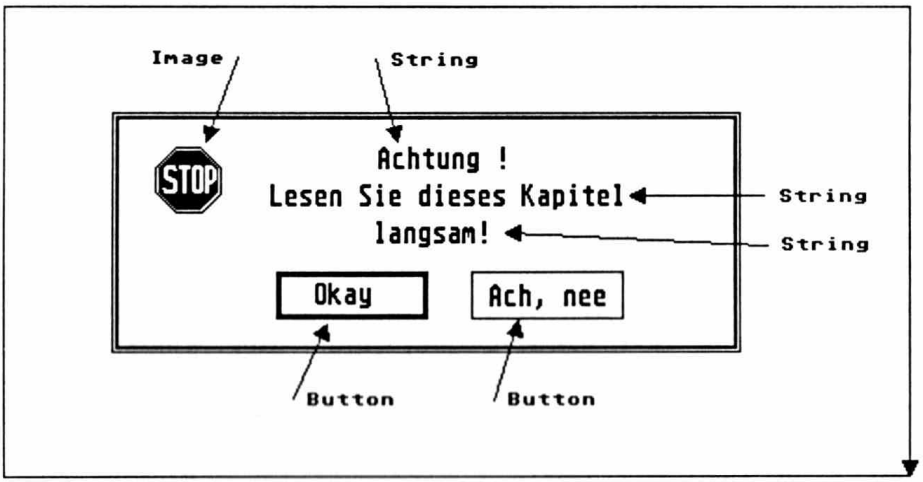
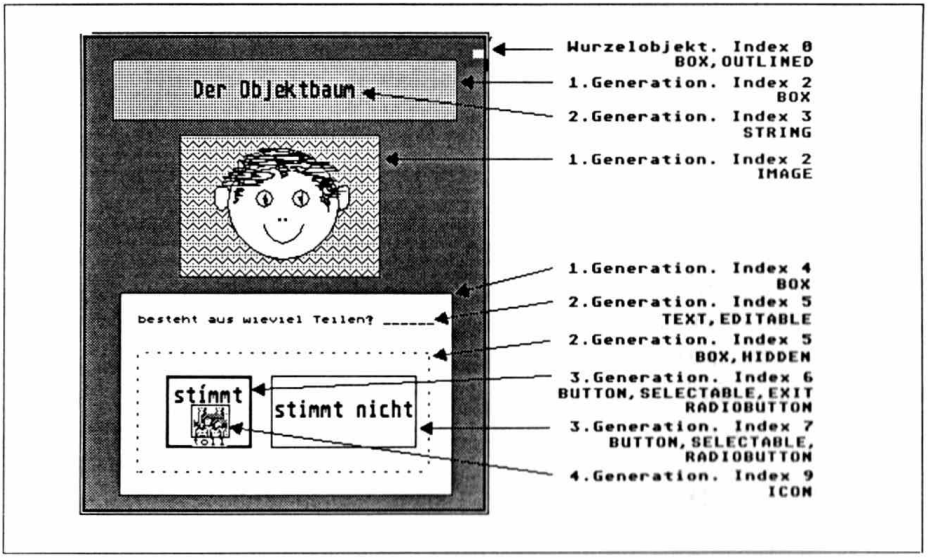


Abb. 9: Objekttypen in der Alertbox

Damit das nicht so trocken stehen bleibt, wird im ersten Listing ein Objekt, bestehend aus der Wurzel und drei Unterobjekten, direkt unter GFA-BASIC aufgebaut. Das ist übrigens ein Verfahren, das seine Berechtigung hat, auch wenn im weiteren viel vom Resource Construction Set die Rede sein wird. Denn ein ausgelagertes Resources-File nimmt, unabhängig von seiner Größe, über 30 KByte weg. Schauen Sie nur einmal, wie wenig im Vergleich dazu unser folgendes Prograämmchen braucht.



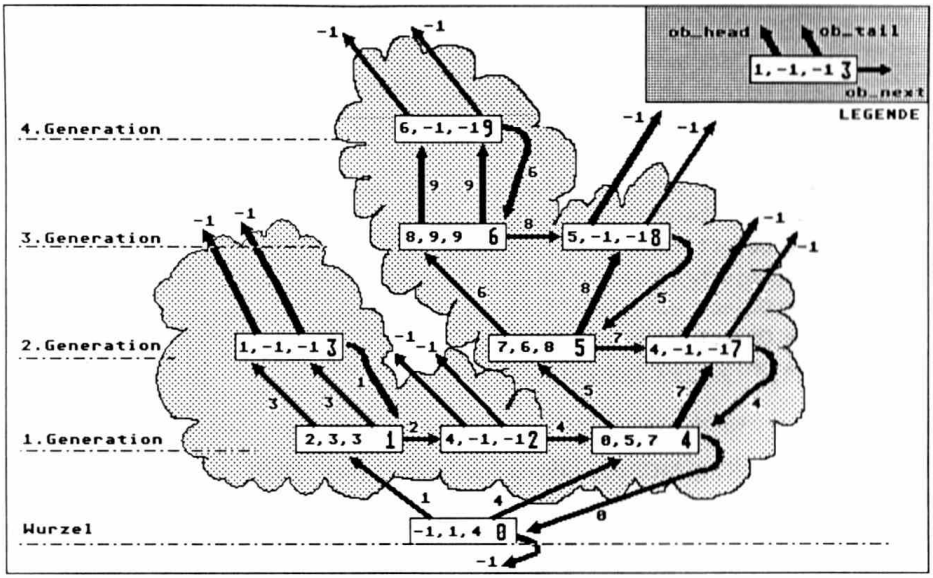


Abb. 10: Objektbaum mit seinen Zeigern

```
Object$=Space$(100*24)
Tedinfo$=Space$(100*28)
'
Object_adresse%=Varptr(Object$)
Tedinfo%=Varptr(Tedinfo$)
```

Zuallererst werden Speicherplätze für die Objektstrukturen reserviert: 24 Bytes * Objektanzahl für die Objekte. Der Anfang dieser Speicherplätze wird durch <Varptr> erfragt. Für die Sonderstruktur TEDINFO, in die bei Objekten, die Text beinhalten, wichtige Informationen abgelegt werden müssen, geschieht genau dasselbe, nur daß diese Struktur jeweils 28 Byte lang sein muß.

4.1.1 Die OBJECT-Struktur

```
' Aufbau und Inhalt der einzelnen Strukturen
' erstes Objekt:
'
Dpoke Object_adresse%,-1
Dpoke Object_adresse%+2,1
Dpoke Object_adresse%+4,3
Dpoke Object_adresse%+6,20
Dpoke Object_adresse%+8,0
Dpoke Object_adresse%+10,0
Gosub Def_obspec(0,2,1,1,1,1)
Lpoke Object_adresse%+12,Ob_spec
Dpoke Object_adresse%+16,0
Dpoke Object_adresse%+18,0
Dpoke Object_adresse%+20,250
Dpoke Object_adresse%+22,200
'
```

In die Objektspeicherplätze werden also die entsprechenden Objektdaten hineingepoked. Dabei muß folgende Reihenfolge gelten:

1. Word (Ob_head)

Die Objektnummer des nächsten Objekts derselben Hierarchie. Wichtig: Die Nummern oder Indizes beginnen bei 0. Es gelten zwei Sonderfälle: Das Wurzelobjekt hat eine -1 (zeigt also auf

die Applikation zurück), alle anderen Objekte erhalten für den Fall, daß kein "Bruder" oder keine "Schwester" vorhanden ist, hier die Nummer des jeweiligen "Vaters", zeigen also auf die Vorgängerhierarchie zurück (siehe Abb. 2).

2. Word (Ob_next)

Die Nummer des ersten Kindes. Falls keins vorhanden ist, wird -1 eingetragen (unser erstes Kind hat also die Nummer 1).

3. Word (Ob_tail)

Die Nummer des letzten Kindes. Falls nur eins da ist, sind Word 2 und Word 3 identisch, falls kein Kind vorhanden ist, kommt auch hier eine -1 hinein (unser letztes Kind hat den Index 3).

4. Word (Ob_typ)

Objekttypschlüssel. GEM hat Zahlen von 20 bis 32 für 12 verschiedene Arten von vordefinierten Objekten reserviert. Dabei handelt es sich um einfache Rechtecke, editierbare Texte, um nicht editierbare Texte, um ICONs oder IMAGES usw.

20	*G_BOX*	eine Pbox mit verschiedenen Rand- und Mustervarianten
21	*G_TEXT*	ein Grafiktext mit TEDINFO-Struktur
22	*G_BOXTEXT*	Grafiktext in einer PBox
23	*G_IMAGE*	Bitmustergrafik mit BITBLK-Struktur
24	*G_PROGDEF*	selbsterzeugte Objektprozedur
25	*G_IBOX*	eine Box, also ohne Muster
26	*G_BUTTON*	Pbox mit normalem Text
27	*G_BOXCHAR*	Pbox mit einem einzigen Buchstaben
28	*G_STRING*	normaler Text
29	*G_FTEXT*	formatierbarer Grafiktext

30	*G_FBOXTEXT*	Pbox mit formatierbarem Grafiktext
31	*G_ICON*	Icon mit Maske, Text und ICONBLK-Struktur
32	*G_TITLE*	Grafiktext für Menüeinträge

5. Word (Ob_flags)

Schlüssel für Objekteigenschaften. Jedes dieser Objekte kann verschiedene Flags setzen, in denen bestimmt wird, ob das Objekt anwählbar und invertierbar oder editierbar oder mit einer Ausstiegsbedingung usw. versehen wird. Wir haben mit der 0 kein Flag gesetzt, das Objekt hat also keinerlei besondere Spezifikation.

0	*NORMAL*	keine Eigenschaft
1	*SELECTABLE*	auswählbar, invertierbar
2	*DEFAULT*	automatische Selektierung durch Drücken der RETURN-Taste
4	*EXIT*	beim Auswählen wird der Dialog automatisch beendet
8	*EDITABLE*	der zugehörige Text kann bearbeitet werden
16	*RBUTTON*	mehrere Objekte im selben Baum lösen sich gegenseitig aus. Es kann also immer nur einer dieser Radio-Buttons selektiert sein.
32	*LASTOB*	das letzte Objekt innerhalb des Baumes muß diesen Flag zeigen
64	*TOUCHEXIT*	wie Exit, Ausstieg aber schon beim Drücken und nicht erst beim Loslassen der Maustatste
128	*HIDETREE*	alle Kinder und Kindeskinder dieses Objekts werden von den Bearbeitungsroutinen ausgelassen

6. Word (Ob_state)

Flag für den Objektzustand beim Zeichnen. Hiermit kann bestimmt werden, ob das Objekt invertiert, umrandet, schattiert, heller und damit nicht ansprechbar usw. auftauchen soll. Dieses Flag wird besonders häufig manipuliert werden.

0	*NORMAL*	Normaldarstellung
1	*SELECTED*	invertierte Darstellung
2	*CROSSED*	das Objekt wird mit einem Kreuz ausgestrichen
4	*CHECKED*	das Objekt wird mit einem Häkchen versehen
8	*DISABLED*	das Objekt wird heller dargestellt
16	*OUTLINED*	das Objekt bekommt einen zusätzlichen Rahmen
32	*SHADOWED*	das Objekt bekommt einen Schatten

7. Langword (also 4 Byte) (Ob_spec)

Informationen, die vom Objekttyp abhängig sind. Bei Boxen, Strings usw., all denjenigen Objekten, für die keine weitere Informationsstruktur vorhanden ist, werden hier Daten über Randbreite, Muster des Hintergrundes usw. bzw. der String selber abgelegt. Ansonsten steht hier die Anfangsadresse des Strukturfeldes, in dem die Folgeinformationen festgehalten werden. Näheres siehe weiter unten.

8. und 9. Word (Ob_x und Ob_y)

X% und Y%-Koordinaten des Objekts, relativ zum Wurzelobjekt. Das erste Objekt, also die Baumwurzel, bekommt hier zwei Nullen.

10. und 11. Word (Ob_w und Ob_h)

Breite und Höhe des Objekts in Pixeln.

Die drei anderen Objekte werden jetzt nach demselben Strickmuster wie das erste aufgebaut.

```
' Zweites Objekt:
Add Object_adresse%,24
Dpoke Object_adresse%,2
Dpoke Object_adresse%+2,-1
Dpoke Object_adresse%+4,-1
Dpoke Object_adresse%+6,20
Dpoke Object_adresse%+8,0
Dpoke Object_adresse%+10,&H4+&H20
Gosub Def_obspec(0,1,1,1,4,1)
Lpoke Object_adresse%+12,Ob_spec
Dpoke Object_adresse%+16,20
Dpoke Object_adresse%+18,40
Dpoke Object_adresse%+20,80
Dpoke Object_adresse%+22,80
'
```

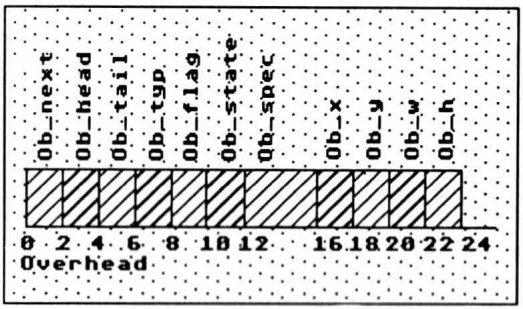


Abb. 11: OBJECT-Struktur

```
' Drittes Objekt:
Add Object_adresse%,24
Dpoke Object_adresse%,3
Dpoke Object_adresse%+2,-1
Dpoke Object_adresse%+4,-1
Dpoke Object_adresse%+6,28
Dpoke Object_adresse%+8,&H20
Dpoke Object_adresse%+10,0
T$="Dies ist ein Test"+Chr$(0)
Ob_spec=Varptr(T$)
Lpoke Object_adresse%+12,Ob_spec
Dpoke Object_adresse%+16,60
Dpoke Object_adresse%+18,10
```

```
Dpoke Object_adresse%+20,17*8
Dpoke Object_adresse%+22,16
'
' Viertes Objekt:
Add Object_adresse%,24
Dpoke Object_adresse%,0
Dpoke Object_adresse%+2,-1
Dpoke Object_adresse%+4,-1
Dpoke Object_adresse%+6,21
Dpoke Object_adresse%+8,&H20
Dpoke Object_adresse%+10,&H8
Lpoke Object_adresse%+12,Tedinfo%
Dpoke Object_adresse%+16,25
Dpoke Object_adresse%+18,150
Dpoke Object_adresse%+20,5*8
Dpoke Object_adresse%+22,16
```

An den Hierarchie-Indices sehen Sie, daß alle Objekte Kinder der ersten sind und selber keine Kinder haben (im 2. und 3. Word steht jeweil eine -1). Das vierte Objekt ist das letzte und weist im ersten Word auf das erste, also seine Wurzel, zurück. Das zweite Objekt ist ebenfalls eine BOX (20), das dritte ein STRING (28) und das vierte ein TEXT (21).

4.1.2 Ob_spec-Varianten

Die Objektspezifikation (also der Inhalt des Longwords) ist schon etwas komplizierter. Beginnen wir mit dem einfachsten: Das STRING-Objekt (unser drittes) bekommt hier den Speicheranfang des Strings, der erscheinen soll, über <Varptr (String$)> mitgeteilt. GEM hat eine generelle Konvention: Alle Strings, die übergeben werden, müssen mit einem Nullbyte enden. Deshalb hier ...+chr$(0).

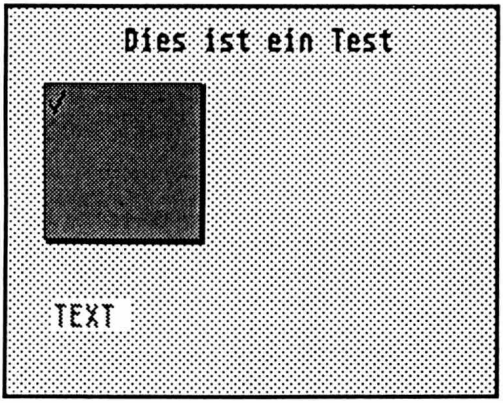


Abb. 12: Der selbsterzeugte Objektbaum

Der BOX und der IBOX werden in Ob_spec Informationen über ein eventuell vorhandenes ASCII-Zeichen, die Randfarbe und -breite, die Textfarbe, das Muster und den Zeichenmodus übergeben. Dabei sind die 32 Bit des Longwords wie folgt belegt:

Setzen und Verändern der Objektspezifikationen

```
+-----------------------------------------------------+
|      | High_word        |  Low_word                 |
| Ob = 00000000 00000011 0001  0001  1 000  0000      |
|      | Zeichen| Rand-   |Rand-|Text-|M|Mus-|Muster-|
|      |        | breite  |farbe|farbe| |ter |farbe  |
+-----------------------------------------------------+
```

In der folgenden Prozedur werden die gewünschten Parameter übergeben, und das Longword wird entsprechend aufgebaut. Das hier benutzte Verfahren splittet die Quelle in Low_Word und High-Word und diese wieder in Low-Byte und High-Byte auf. Im High-Word, also den ersten 16 Bit des Longwords, werden im High_Byte das ASCII-Zeichen und im Low-Byte die Randbreite übergeben. Zusammengeführt werden diese wie bekannt über (High-Byte*256)+Low_byte. Das Low-Word wird im High-Byte noch einmal unterteilt. Die Zusammenführung geschieht mit (High-Halbbyte*16)+Low_Halbbyte. Ebenso beim

Low-Byte des Low-Word. Hier bedienen wir uns zusätzlich noch eines Tricks. Das den Zeichenmodus angebende Bit wird in die Musterbits integriert, so daß Muster von 0 bis 7 transparent als und Muster von 8 bis 15 als überschreibend gelten. Jetzt werden nur noch die beiden Words zusammengeführt: (High-Word*65536)+ Low-Word. Und das Ergebnis kann an das ob_spec der Objektstruktur übergeben werden.

```
Procedure Def_obspec(Char%,Rand%,Randfarbe%,Textfarbe%,
                                        Muster%,Musterfarbe%)
  ' Char%:        0 bzw.  ASCII-Wert
  ' Rand%:        0-127 nach innen,128-255 nach außen dicker werdend
  ' Randfarbe%:   1 oder 0
  ' Textfarbe%:   1 oder 0
  ' Muster%:      0-7 transparent, 8-15 replace
  ' Musterfarbe%: 1 oder 0
  Ob_spec=((Char%*256+Rand%)*65536)+
            ((Randfarbe%*16+Textfarbe%)*256+
                    (Muster%*16+Musterfarbe%))
Return
```

(Die Zeilen sind aus drucktechnischen Gründen geteilt)

Diese Prozedur gehört natürlich weiter nach hinten ins Programm. Hier geht es jetzt nämlich weiter mit den Sonderstrukturen, deren Adressen an ob_spec übergeben werden können.

4.1.3 Die TEDINFO-Struktur

Insgesamt gibt es davon vier: eine für editierbare Texte, die TEDINFO-Struktur, eine für Icons, die ICONBLK-Struktur, eine für Bilder, die als Objekte bedienbar sein sollen, die BITBLK-Struktur und eine, die in benutzereigene Routinen verspringt, wenn ein Objekt angeklickt wird, die APPLBLK-Struktur. In diesem Kapitel wird TEDINFO vorgestellt.

```
Rs_string1$="TEXT"+Chr$(0)
Rs_string2$=""+Chr$(0)
Rs_string3$=""+Chr$(0)
'
```

Auch hier wird für den Text, der auf dem Bildschirm erscheinen soll, Speicherplatz geschaffen. TEDINFO will aber drei Strings übergeben bekommen, damit später das Editieren des Textes innerhalb gewisser Konventionen erfolgen kann. Einen (den ersten) für die Textmaske. Das ist der Text, der bei Editionen stehen bleibt, z.B. bei einem Adressenformular: Name: _____.

Der zweite String gibt Eingabegültigkeiten an, Sie können z.B. bestimmen, ob nur Zahlen, nur Großbuchstaben usw. eingegeben werden dürfen: ~~~~~~XXXXXXX

Dabei stehen:

X alle Zeichen gelten
9 nur Zahlen gelten
P alle Zeichen, die auch in Datei- und Pfadnamen gelten
A nur Großbuchstaben
a nur Kleinbuchstaben
N Großbuchstaben und Zahlen
n Großbuchstaben, Kleinbuchstaben und Zahlen

In den dritten String werden beim Editieren die Eingaben eingeschrieben, Sie können diesen leerlassen, Sie können aber auch eine Vorgabemaske wählen. Auch hier die Konvention des Nullbytes am Ende des Strings: ~~~~~_____

```
 Lpoke Tedinfo%,Varptr(Rs_string1$)
Lpoke Tedinfo%+4,Varptr(Rs_string2$)
Lpoke Tedinfo%+8,Varptr(Rs_string3$)
Dpoke Tedinfo%+12,3
Dpoke Tedinfo%+14,6
Dpoke Tedinfo%+16,0
Dpoke Tedinfo%+18,&H1180
Dpoke Tedinfo%+20,0
Dpoke Tedinfo%+22,-1
Dpoke Tedinfo%+24,5
Dpoke Tedinfo%+26,1
```

Dieses Speicherfeld wird im Prinzip genauso behandelt wie die Objektstruktur. Mehrere TEDINFOs brauchen also hintereinander liegenden Platz. Einträge hier ebenfalls durch Lpoke- und Dpoke-Befehle.

1. Longword (Te_ptext)

Die Anfangsadresse des Textes

2. Longword (Te_ptmplt)

Die Anfangsadresse der Textmaske

3. Longword (Te_pvalid)

Die Anfangsadresse der Gültigkeitsangaben

4. Word (Te_font)

Der Zeichensatz, mit dem geschrieben werden soll. Es stehen derzeit nur zwei Indizes zur Verfügung: 3 für Normalgroße und 6 für kleine Schrift.

5. Word (Te_junk1)

Ist reserviert, muß immer eine -1 enthalten.

6. Word (Te_just)

Justage des Textes, also ob linksbündig (1), zentriert(2) oder rechtsbündig(3).

7. Word (Te_color)

Farbinformationen, entsprechend dem Low-Longword des ob_spec, das oben erläutert wurde. Für Text und Schwarz-Weiß-Monitor ist &H1180 angemessen.

8. Word (Te_junk2)

Ebenfalls reserviert, hier hat eine -1 zu stehen

9. Word (Te_thickness)

Dicke des Randes bei BOXTEXT. Als Zahl zwischen - 127 und +127.

10. Word (Te_txtlen)

Länge des Textstrings(des ersten) in Buchstaben, incl. des 0-Bytes.

11. Word (Te_tmplen)

Länge des Maskenstrings in Buchstaben (ebenfalls incl. des 0-Bytes)

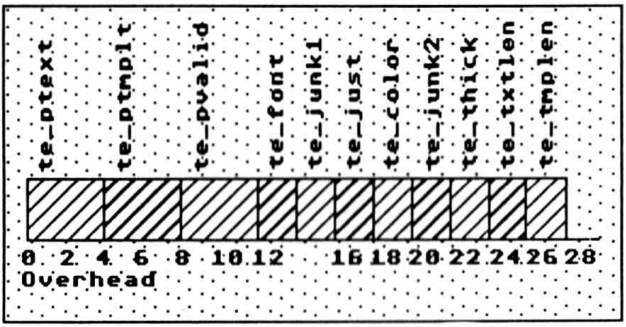


Abb. 13: TEDINFO-Struktur

So. Damit haben Sie einen kompletten Objektbaum eingegeben und können GEM befehlen, diesen auf den Bildschirm zu malen.

4.2 Objekte zeichnen sich selbst

In der überaus ausführlichen Routinenbibliothek von GEM stehen natürlich fertige Pakete zum Zeichnen von Objekten und Objektbäumen zur Verfügung. Mit <Objc-draw> wird all das, was Sie eben entworfen haben, auf einen Schlag auf den Bildschirm geschrieben. Ja, es geht noch komfortabler: Damit Sie nicht erst ausrechnen müssen, welche Koordinaten Ihr Objektbaum benötigt, um mittig auf dem Bildschrim zu erscheinen, benutzen Sie die Routine <Form-Center>, die Ihnen genau das abnimmt.

Springen Sie jetzt die folgende Prozedur an, die beide Routinen beinhaltet. Als Parameter will die Adresse des Baumes übergeben werden sowie das erste zu zeichnende Objekt (hier die Wurzel) und die letzte zu zeichnende Ebene (hier haben wir nur zwei, nämlich die Wurzel und die restlichen als Kinder ohne Kindeskinder). Dieser Wert ist aber unkritisch.

```
Gosub Objc_draw(Object_adresse%,0,2)
Do
  Exit If Mousek
Loop
End
```

Und Sie staunen: Ihr erster Objektbaum aus vier verschiedenen GEM-Objekten steht auf dem Bildschirm.

```
' ****************************************
' * Library-Routine                      *
' Procedure Objc_draw(Adr%,01%,02%)
Lpoke Addrin,adr%
Gemsys 54
Dpoke Gintin,01%
Dpoke Gintin+2,02%
Gemsys 42
Return
```

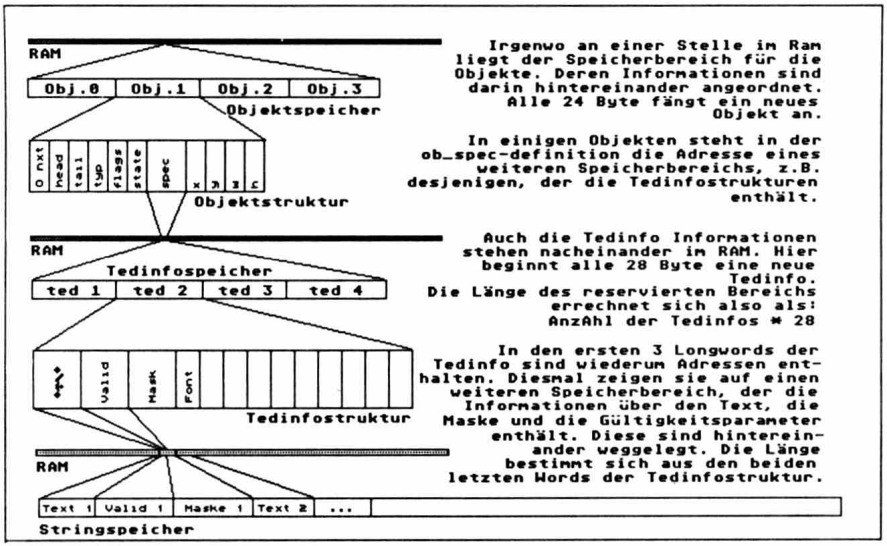


Abb. 14: Ihr erster Ojektbaum

4.3 Resource Construction Set-Spielereien

Nachdem der Selbstaufbau von Objektstrukturen so schön funktioniert hat, soll natürlich nicht verschwiegen werden, daß das manuelle Konstruieren von Objektbäumen eine Heidenarbeit ist. Aber es gibt ja das RCS, das Resource Construction Set (z.B. in der Data Welt-Edition). Wer es noch nicht hat, sollte es sich schleunigst besorgen, denn wir beziehen uns im folgenden darauf. Mit Hilfe des RCS können Sie unter einer grafischen Benutzerführung die Bäume "WYSIWYG", "What You See Is What You Get", aufbauen und die RCS-Files, die "Resources", generieren, die die gesamte AES-Objektstruktur außerhalb des Programms lagern.

Jetzt sollten Sie, um das Folgende nachvollziehen zu können, mit Hilfe des RCS zwei Objektbäume einrichten:

Baum Nr.1

Benutzen Sie die DIALOG-Option, und nennen Sie den Baum <Tree1>. Dahinein kommt ein <EDIT: _____>-Objekt, das folgendermaßen definiert wird:

```
<PTMPLT>Ich bin ~~>
<PVALID>~~~~~~~~99>
<PTEXT >~~~~~~~~__>
```

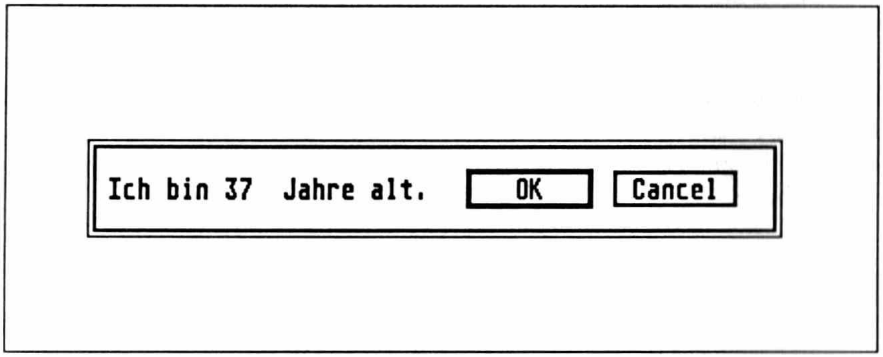


Abb. 15: Baum Nr. 1

Dahinter ordnen Sie einen <STRING> an: "Jahre alt". Zwei <BUTTONS>, mit "Cancel" und "OK" beschriftet, beide mit <EXIT>, <SELECTABLE> und <RADIO BUTTON>-Flag, vervollständigen diesen Baum. Jetzt benennen Sie den <EDIT>-Text mit dem Namen "T1TXT", den "OK"-Button mit "T1OK" und den "Cancel"-Button mit "T1CAN".

Baum Nr. 2

Dieser Baum besteht aus einem <EDIT>- und vielen <BUTTON>-Objekten. Weil Sie ja nur ein einziges Alter haben, müssen die Jahreszahlen-Buttons mit dem <RADIO-BUTTON>-Flag versehen werden, genauso wie "OK" und "Cancel". Das "Radio" bei den Radiobuttons bezieht sich aber immer auf alle (!) Objekte innerhalb einer Hierarchie. Deshalb muß also unter

eine der beiden Radio-Strukturen eine <HIDDEN>-Box gelegt werden. Erst dann arbeiten sie unabhängig voneinander.

Das Beispiel geht davon aus, daß der FREE- oder DIALOG-Baum "Tree2" heißt, die 10 Knöpfe als <BUTTON> dargestellt sind, alle <SELECTABLE>, <RADIO BUTTON> und <EXIT> angewählt haben und die "OK" und "Cancel"-Knöpfe sowie der <EDIT>-Text wie im Tree1 aufgebaut sind.

Benennen Sie die Zahlenbuttons mit "T21" bis "T210", den Text mit "T2TXT" und die Exit-Knöpfe mit "T2OK" bzw. "T2CAN".

Jetzt brauchen Sie nur noch im RCS-Menütitel "GLOBAL" unter "OUTPUT" die Option "C-SOURCE" anklicken, weil aus diesem Source einige Daten benötigt werden, und unter "DEMO.RSC" abspeichern.

Abb. 16: Baum Nr. 2

Das RCS spuckt vier Files aus: die Resource, die Sie später ins Programm einlinken müssen, einen *.DEF-File, der nur vom RCS wiederverwendet wird, die C-Source der Objektstruktur sowie einen Header-File "DEMO.H", der die Objektindizes den

von Ihnen eingegebenen Namen zuweist. Dieser File ist natürlich auf den C-Compiler zugeschnitten und muß in seiner Syntax geändert werden, damit GFA-BASIC ihn versteht. Aus "# DEFINE TREE1 0 /*TREE */" usw. muß werden "TREE1%=0 !*TREE */" usw.

4.3.1 Header-Files aus dem RCS nach BASIC konvertieren

Im folgenden finden Sie ein kleines Programm, das Ihnen die Arbeit des Umschreibens abnimmt. Wenn Sie auch von BASIC reservierte Worte als Variablennamen benutzen wollen, sollten Sie die REM-Zeile ohne REM übernehmen. Dann wird als zweites Variablennamenzeichen ein Punkt eingefügt. Beispiel: Aus GOTO wird G.OTO.

```
' Programm: RCS_HEAD.BAS
'
' ***************************************************
' *  Umwandlung des *.H-Files vom RCS in GFA-BASIC *
' ***************************************************
'
Do
  Print At(1,1);"Bitte den umzuwandelnden  *.H -File auswählen
                                          oder Abbruch anklicken "
  Fileselect "A:\*.H","",File$
  If File$<>""
    Gosub Do_it
  Else
    End
  Endif
Loop
'
Procedure Do_it
  Open "I",#1,File$
  File2$=Left$(File$,Instr(File$,"."))+"lst"
  Open "O",#2,File2$
  Repeat
    Input #1,A$
    A$=Mid$(A$,9)
    A$=Left$(A$,1)+"."+Mid$(A$,2)          ! Löschen Sie das ",
    '                                      ! falls Sie reservierte
    '                                      ! Worte benutzen wollen
    A%=Instr(A$," ")
```

```
      A$=Left$(A$,A%-1)+"%="+Mid$(A$,A%+1)
      A%=Instr(A$,"/")
      Mid$(A$,A%-1,1)="!"
      Print A$
      Print #2,A$
      Inc N%
    Until Lof(#1)-1=Loc(#1)
    Close #1
    Close #2
  Return
```

Ja, das war's schon. Auch kurze Listings bringen den gewollten Effekt. Es passiert folgendes: Nachdem der Filename erfragt wurde, kann die dazugehörige Datei im Lesemodus geöffnet werden, eine zweite Datei mit demselben Namen, aber der Endung <*.LST> wird im Schreibmodus geöffnet. In einer Schleife mit der ungewohnten Ausstiegsbedingung <Lof(#1)-1=Loc(#1), die das Ende des Files anzeigt, wird Zeile für Zeile eingelesen, um die ersten 8 Zeichen (#define)gekürzt und nach dem Leerzeichen zwischen Konstantenname und Index gesucht. Dieses Leerzeichen wird durch "% =" ersetzt. Des weiteren wird nach dem Kommentarzeichen von C, dem Backslash "\" gesucht, der durch das Kommentarzeichen von BASIC, "!", ersetzt wird. Auf den Bildschirm sowie in die Datei schreiben beendet die Schleife. Aufräumen (soll heißen Dateien schließen) beendet den Durchgang.

Das Ergebnis des Programmlaufs, den File "DEMO.LST", mergen Sie jetzt ins BASIC-Programm. Aus dem C-Source können Sie noch folgende Zeilen übernehmen, die das Leben nachher einfacher machen:

```
# define T1OBJ 0          =     T1_begin%=0
# define T2OBJ 5          =     T2_begin%=5
```

Diese Werte geben jeweils den Anfangsindex der Bäume innerhalb des RSC-Objekt-Arrays an. Das folgende Beispiel finden Sie unter dem Namen "FORMULAR.BAS" auf der Diskette.

```
T1_begin%=0
Tree1%=0        !/* TREE */
T1_txt%=1       !/* OBJECT in TREE #0 */
T1_ok%=3        !/* OBJECT in TREE #0 */
T1_can%=4       !/* OBJECT in TREE #0 */
T2_begin%=5
Tree2%=1        !/* TREE */
T2_txt%=1       !/* OBJECT in TREE #1 */
T2_b1%=4        !/* OBJECT in TREE #1 */
T2_b2%=5        !/* OBJECT in TREE #1 */
T2_b3%=6        !/* OBJECT in TREE #1 */
T2_b4%=7        !/* OBJECT in TREE #1 */
T2_b5%=8        !/* OBJECT in TREE #1 */
T2_b6%=9        !/* OBJECT in TREE #1 */
T2_b7%=10       !/* OBJECT in TREE #1 */
T2_b8%=11       !/* OBJECT in TREE #1 */
T2_b9%=12       !/* OBJECT in TREE #1 */
T2_b10%=13      !/* OBJECT in TREE #1 */
T2_ok%=14       !/* OBJECT in TREE #1 */
T2_can%=15      !/* OBJECT in TREE #1 */
```

Wenn die Indizes bei Ihnen etwas anders aussehen, macht's nichts, solange die Struktur stimmt.

Jetzt sind die Vorbereitungen soweit gediehen, daß das RSC-File geladen werden müßte und Sie endlich etwas über Ihr Alter erfahren.

4.4 Resources-Files laden und freigeben

Es muß jetzt der RSC-File in den Arbeitsspeicher geladen werden. Zuständig dafür ist die AES-Routine <Rsrc_load>. In BASIC wird sie aufgerufen mit <Gosub Rsrc_load("DEMO.RSC")>. Diese Prozedur beinhaltet eine kleine Fehlerbehandlung, indem eine Alertbox bei Ladehemmungen darauf hinweist, den Speicher wieder freigibt und das Programm beendet.

```
! ********************
! * Library Routinen *
!
Procedure Rsrc_load(R$)
  Lpoke Addrin,Varptr(R$)
  Gemsys 110
  If Dpeek(Gintout)=0
    Alert 3,"Es konnte kein RSC-File|geladen werden",1,"Ende",k%
    Gosub Rsrc_free
    End
  Endif
Return
```

Ab jetzt ist eine eindringliche Warnung angebracht: RSC-Files werden speicherresident gelagert. Das heißt, wenn dieser Speicherbereich nicht wieder ordnungsgemäß freigegeben wird, knabbert jeder Neustart an der Basepage und irgendwann - mit Sicherheit gerade dann, wenn Sie die letzten Änderungen noch nicht gesichert haben, werden Sie nur noch einen Systemzusammenbruch konstatieren können.

Also: im Verlauf der Programmierung

- Ihre Schreibereien immer wieder abspeichern.
- Immer mit Hilfe des Programms abbrechen.
- Falls das nicht geht, im Direktmodus eingeben: gosub rsrc_free

Deshalb gleich im Anschluß die Prozedur, die den Speicherbereich wieder freigibt. Sie wird ohne Parameter mit <Gosub Rsrc_free> aufgerufen:

```
! *******************
! * Library Routine *
!
Procedure Rsrc_free
  Gemsys 111
Return
```

Da liegt also nun das RSC irgendwo im freien Speicher. Oben in unserem Objekt-Baukasten haben wir die Lage über den

<Varptr> erfragen können. AES dagegen stellt uns eine Routine zur Verfügung, mit deren Hilfe wir alle benötigten Baum- und Objectadressen, also deren genaue Lage im Speicherraum, erfragen können. Die <Procedure Rsrc_gaddr> gibt je nach Parameter entweder die Anfangsadresse einer Baumstruktur, die Anfangsadresse eines Objekts innerhalb eines Baumes, die Adresse einer TEDINFO-Struktur oder auch andere Adressen zurück:

```
!********************
! * Library Routine *
!
Procedure Rsrc_gaddr(F%,O%,S%)
  Dpoke Gintin,F%
  Dpoke Gintin+2,O%
  Gemsys 112
  *S%=Lpeek(Addrout)
Return
```

Mit <Gosub Rsrc_gaddr(Flag%,Objektindex%,*Adresse)> wird die Funktion aufgerufen. Der Flag% selektiert die Struktur, deren Adresse Sie haben wollen:

0 Adresse eines ganzen Baumes
1 Adresse eines Objekts innerhalb eines Baumes
2 Anfangsadresse der TEDINFO-Struktur
3 Anfangsadresse der ICONBLK-Struktur
4 Anfangsadresse der BITBLK-Struktur
5 Textstringadresse
6 Bitmusterdatenadresse
7 Zeiger Ob_spec aus der OBJECT_Struktur
8 Zeiger Te_ptext aus der TEDINFO-Struktur
9 Zeiger Te_ptmplt aus der TEDINFO-Struktur
10 Zeiger Te_pvalid aus der TEDINFO-AStruktur
11 Zeiger Ib_pmask aus der ICONBLK-Struktur
12 Zeiger Ib_pdata aus der ICONBLK-Struktur
13 Zeiger Ib_ptext aus der ICONBLK-Struktur
14 Zeiger Bi_pdata aus der BITBLK-Struktur

Objektindex% erwartet den Konstantennamen aus der <Procedure Rsc_data>, der auf die gesuchte Struktur verweist. Aber Achtung: Bäume werden direkt mit dem Index adressiert,

d.h., Sie geben ein: "Tree1%" bzw. "Tree2%" usw. Einzelobjekte dagegen wollen absolut angesprochen werden. Deshalb ist hier der Objektindex% immer: Anfangsindex des Baumes + relativer Objektindex, also z.B. "T1_begin%+T1_txt%". Und Tedinfo-Strukturen werden nach der Reihenfolge innerhalb ihres eigenen Feldes angesprochen (dazu später).

Unser Beispielprogramm kann jetzt gestartet werden: Der RSC-File wird geladen, und die Anfangsadressen der beiden Bäume innerhalb des freien Speichers werden ermittelt.

```
Gosub Rsrc_load("demo.rsc")
Gosub Rsrc_gaddr(0,Tree1%,*Adr1)
Gosub Rsrc_gaddr(0,Tree2%,*Adr2)
```

Wenn Sie keine Geduld haben, können Sie schon mal versuchen:

```
Gosub Objc_draw(Adr1,0,2)
If Mousek=1
  cls
  Gosub Objc_draw(Adr2,0,3)
Endif
Do
  Exit if Mousek=2
Loop
Gosub Rsrc_free
End
```

4.5 Statusfragen

Die Objektboxen sind - was ihre inhaltliche Bedeutung angeht - jungfräulich. Um Aussagen machen zu können oder Ergebnisse zu erfragen, werden Vorbelegungungen und Zustandsabfragen benötigt.

Es soll z.B. der Maskentext in Tree1% mit dem Alter 37 Jahre vorbelegt werden, der zuständige String muß also angesprochen werden, und in Tree2% soll der Button "35 Jahre" beim Zeichnen selektiert (invertiert) sein, das Statusflag also auf 1 gesetzt werden.

Für Statusfragen findet sich eine fertige AES-Routine:

```
! ********************
! * Library-Routine *
!
Procedure Objc_change(A,O%,S%,X%,Y%,W%,H%,F%)
  Dpoke Gintin,O%
  Dpoke Gintin+4,X%
  Dpoke Gintin+6,Y%
  Dpoke Gintin+8,W%
  Dpoke Gintin+10,H%
  Dpoke Gintin+12,S%
  Dpoke Gintin+14,F%
  Lpoke Addrin,A
  Gemsys 47
Return
```

Sie wird aufgerufen mit:

```
Gosub Objc-change(Adresse,Index%,Neuer_status%,Koordinaten,Flag%)
```

4.5.1 Ob_state

Die Parameter Adresse und Index% beziehen sich wie immer auf Baumadresse und anzusprechendes Objekt (hier wird der relative Index verwendet, weil ja die Baumadresse schon ausreichend referenziert), die Koordinaten definieren ein Rechteck, innerhalb dessen Grenzen die Statusänderungen erfolgen. Sie könnten damit auch einen halben Button invertieren. Im Normalfall übergeben Sie entweder die Koordinaten der Wurzel (siehe später) oder die des Bildschirms. Im Parameter Neuer_status% können folgende Werte (oder auch Kombinationen davon) übergeben werden:

0	*(NORMAL)*	Normaldarstellung
1	*(SELECTED)*	Invertiert, Selektiert
2	*(CROSSED)*	Durchkreuzt
4	*(CHECKED)*	Das Objekt wird mit einem Häkchen versehen.

8	*(DISABLED)*	Das Objekt wird heller gezeichnet.
16	*(OUTLINED)*	Das Objekt bekommt eine zusätzliche Umrahmung.
32	*(SHADOWED)*	Das Objekt wird schattiert dargestellt.

Der 'letzte Parameter (Flag%) gibt an, ob das Objekt neu gezeichnet werden soll (1) oder nicht (0).

In unserem Beispiel schreiben Sie jetzt vor die Zeile <Gosub Objc-Draw(...)> folgendes:

```
Gosub Objc_change(Ad2,T2_b5%,1,0,0,640,400,0)
```

An die editierbaren Strings heranzukommen, wird schon aufwendiger. Hier wird ein direkter Eingriff in die Objektstrukturdaten erforderlich. Speziell in die TEDINFO-Struktur.

Folgende Schritte sind nötig, um in diese Adresse einen Text einzuschreiben:

1. Ermitteln der Adresse, an der die TEDINFO-Struktur beginnt.

   ```
   Gosub Rsrc_gaddr(2,Tedinfo_index%,*Tedinfo_adr)
   ```

   Hier wird im zweiten Parameter der Index des jeweiligen Tedinfo-Textes erwartet. Wir haben zwei davon, einen in Tree1% und einen in Tree2%. Auch dieses Feld beginnt mit 0. Da hier also der erste geändert werden soll, heißt der Parameter "0".

2. Nacheinander in die so ermittelte Adresse die einzelnen Buchstaben des Strings einschreiben.

   ```
   For N%=0 to Len(Text$)
     Poke Tedinfo_adr+N%,Mid$(Text$,N%,1)
   Next N%
   ```

   Daraus machen wir aber eine verallgemeinerte Library-Funktion mit folgender Aufrufstruktur:

```
<Gosub Obtxt_set(Tedinfoindex%,Textlänge%,Text$)>.

! ******************
! * Library-Routine *
!
Procedure Obtxt_set(O%,L%,T$)
  Local Ad,Ted,N%,A%
  Gosub Rsrc_gaddr(2,O%,*Ad)
  Ted=Lpeek(Ad)
  For N%=0 To L%-1
    A%=Asc(Mid$(T$,N%+1,1))
    Poke Ted+N%,A%
  Next N%
Return
```

Jetzt können Sie oben in das Programm (ebenfalls vor <Gosub Objc-Draw(...)>) eine weitere Zeile setzen:

```
Gosub Obtxt_set(0,2,"37")
```

Wo wir gerade so weit sind, können wir das Verfahren ja auch umdrehen und damit die Routine bauen, mit der ein Text innerhalb der TEDINFO-Struktur gelesen werden kann. Wenn Sie inzwischen mit der Pointer-Hangelei vertraut sind, werden Sie kein großes Fragezeichen mehr vor Augen haben, wenn jetzt aus dem Longword der TEDINFO-Struktur die Textadresse herausgepeekt und aus dieser Adresse Byte für Byte der ASCII-Wert gelesen wird.

```
Procedure Obtxt_get(O%,L%,S)
  Local Ad,N%,T$
  Gosub Rsrc_gaddr(2,O%,*Ad)
  For N%=0 To L%
    T$=T$+Chr$(Peek(Lpeek(Ad)+N%))
  Next N%
  *S=T$
Return
```

Übergeben wird wieder der Index der jeweiligen TEDINFO-Struktur und die Länge des abzufragenden Strings. Zurückgegeben wird die Stringadresse, deren Inhalt im GFA-BASIC bekanntlich direkt erfragt werden kann.

Und zur Vervollständigung - weil sie ebenfalls häufig gebraucht wird - die Prozedur, die das Statusflag des Objekts ausliest, also das Gegenstück zu <Objc_change>. Sie wissen inzwischen, daß das 5. Word dieses Flag beinhaltet. Es muß also lediglich die Anfangsadresse des Objekts innerhalb seiner Objektstruktur herausgefunden und dann 10 Byte weiter der Inhalt des Longwords gepeekt werden.

```
Procedure Obstate_get(O%,S%)
  Local Ad
  Gosub Rsrc_gaddr(1,O%,*Ad)
  *S%=Dpeek(Ad+10)
Return
```

Übergeben wird wie üblich der Objektindex, hier muß es der absolute Index innerhalb der Gesamtstruktur sein. Deshalb der Aufruf z.B. mit <Tree_begin%+Ob_index%>. In S% wird der jeweilige Status zurückgegeben, wobei dieselben Zahlenwerte wie in Objc-change beschrieben herauskommen. Zu beachten ist, daß diese sich addieren können, d.h., ein Objekt mit SELECTED-Flag und OUTLINED-Flag hat den Wert &H11.

Jetzt wischen Sie sich erst einmal den Schweiß von der Stirn.

4.6 Formulare

Bisher haben Sie immer von Objekten oder Objektbäumen gehört. Digital Research hat für Dialogbäume den Begriff "Formular" eingeführt, der meines Erachtens hervorragend paßt. Denn ein Formular ist nichts anderes als ein standardisiertes Medium für Fragen und Antworten. Angefangen beim Lottoschein bis hin zur Steuererklärung haben alle diese Formulare gemeinsam, daß an bestimmten Stellen bestimmte Eingaben erwartet werden. Und ein Formular in GEM ist ein vordefiniertes Rechteck, in dem mit Hilfe der Maus etwas selektiert wird oder mit Hilfe der Tastatur bestimmte Eingaben erwartet werden.

Um diese Bedienung auch realisieren zu können, steht eine AES-Routine zur Verfügung, die Sie für das ganze Brimborium um Strukturen, Prozeduren, Pointer und Bäume entschädigt: <Form-Do>.

```
' *****************
' Library_Routine *
Procedure Form_do(A,O%,S%)
  Dpoke Gintin,O%
  Lpoke Addrin,A
  Gemsys 50
  *S%=Dpeek(Gintout)
Return
```

In diesen 6 Zeilen ist das komplette Formularhandling untergebracht. Es ist kaum zu glauben, was die Digital Research-Programmierer geleistet haben: Solange, bis ein EXIT-Objekt angeklickt wurde, verbleibt das Programm in dieser Routine. Der Benutzer kann Boxen selektieren und invertieren, Radio-Boxen lösen sich selbsttätig gegeneinander aus, in die vorformulierten Edit-Strings kann der mit der Maske und den Gültigkeitsparametern vordefinierte Text eingeben werden, wobei die üblichen Tasten in Funktion bleiben: ESC löscht den String, Delete und Backspace löschen je ein Zeichen, und die Pfeiltasten funktionieren auch. Zurück erhalten Sie den Index desjenigen Objekts, das für die Rückkehr verantwortlich war. Und das alles in 6 BASIC-Zeilen.

Die Prozedur wird aufgerufen mit

```
<Gosub Form_do(Baumadresse,Objektindex%,*Return-Objekt%)>
```

Der Parameter <Objektindex%> wird nur beim Vorhandensein von Edit-Objekten oder Text-Objekten belegt, dann aber mit dem relativen Objektindex des ersten Textstrings dieses Baumes. Soll kein Maskentext durch die Routine angesprochen werden, sondern nur Boxen o.ä., wird eine 0 übergeben.

Wenn Sie jetzt im Programm hinter den <Gosub Objc_draw(...)>-Aufruf folgende Zeilen setzen, können Sie zwei Zahlen als Altersangabe eingeben, evtl. korrigieren, mit dem EXIT-Button aussteigen, das Objekt wieder löschen, die

Objc_change-Prozedur setzt das durch das Anklicken selektierte Exitobjekt wieder auf Normal zurück, und Ihr Alter steht auf dem Bildschirm.

```
Gosub Form_do(Ad1,T1_txt%,*Ret_ob%)
Gosub Objc_change(Ad1,Ret_ob%,0,0,X,Y,W,H)
Gosub Obtxt_get(0,2,*Txt$)
Print TxT$
```

In unserer zweiten Demo-Box ist etwas mehr an Abfrage- und Eingabemöglichkeiten eingebaut. Ich zeige, wie erreicht werden kann, daß nach "Canceln" der alte Zustand wieder erscheint, wie das Ergebnis des angeklickten Buttons in den Maskentext geschrieben wird und wie der Status der Buttons überhaupt abgefragt werden kann:

```
Procedure Formular_bedienung

  Sget Screen$                                  ! Retten des Bildschirms
  Gosub Objc_draw(Ad2,0,3)                      ! Zeichen des Objekts
  '
  For N%=T2_begin%+T2_b1% To T2_begin%+T2_b10% ! Erfragen des Status
    Gosub Obstate_get(N%,*State%)               ! der einzelnen Buttons
    Exit If Hex$(State%)="1"                    ! Ausstieg, wenn SELECTED
  Next N%                                       !
  Old_state%=N%-T2_begin%                       ! Selektierter Button
  Gosub Obtxt_get(1,2,*Old_str$)                ! Textinhalt
  '
  Do
    Gosub Form_do(Ad2,0,*Ret_ob%)               ! Formularroutine
    '
    If Ret_ob%<>T2_ok% And Ret_ob%<>T2_can%     ! Bei OK und CANCEL ist
      For N%=T2_begin%+T2_b1% To T2_begin%+T2_b10%  ! die Abfrage nach
        Gosub Obstate_get(N%,*State%)           ! dem Selektierten Button
        Exit If Hex$(State%)="1"                ! nicht nötig
      Next N%
      String$=Str$((N%-1-T2_begin%-3)*5+10)     ! Das Alter
      Gosub Obtxt_set(1,2,String$)              ! wird in das Text-Objekt
      Gosub Objc_draw(Ad2,T2_txt%,1,X,Y,W,H)    ! geschrieben und das neu
    Endif                                       ! gezeichnet
    Exit If Ret_ob%=T2_ok% Or Ret_ob%=T2_can%   ! OK und CANCEL brechen
  Loop                                          ! die Schleife ab
  Gosub Objc_change(Ad2,Ret_ob%,0,0,X,Y,W,H)    ! Normalisieren des
  '                                             ! Button
```

```
  Sput Screen$                                  ! Der alte Bildschirm
  !                                             ! wird wiederhergestellt
  If Ret_ob%=T2_can%                            ! Bei CANCEL wird der
    Gosub Objc_change(Ad2,N%-T2_begin%,0,0,X,Y,W,H) ! alte Zustand
    Gosub Objc_change(Ad2,Old_state%,1,0,X,Y,W,H)   ! wiederhergestellt
    Gosub Obtxt_set(1,2,Old_str$)               ! wie gehabt vor Eintritt
  Endif                                         ! in die Routine.
Return                                          ! Das wars
```

Form_center und Objc_draw sind bekannt. In der folgenden For..Next-Schleife wird das Zustands-Flag der Buttons abgefragt. In unserem Falle (Radio-Buttons) hat nur ein einziger den Status SELECTED, also &H1. Es werden der Reihe nach alle Knöpfe auf diesen Status abgefragt. Die Exit if ..-Bedingung gibt die Möglichkeit, aus dem letzten For..Next-Zähler das jeweilige Objekt zu errechnen.

So. Jetzt wird der alte Status und mit Hilfe von <Obtxt_get> auch der alte String gerettet.

Die Form_do-Routine läuft in einer Do..Loop-Schleife, da ja fast alle Objekte mit EXIT gekennzeichnet sind und beim Anwählen jedesmal die Form-do-Routine verlassen wird. Ahnen Sie schon warum? Form_do ist eine in sich geschlossene Routine, in die nur nach vorgegebenen Mustern etwas eingegeben werden kann. Wollen Sie die Eingabe auswerten und so wie hier z.B. den Wert, den der Button repräsentiert, in ein Edit-Objekt überführen, kann das nur außerhalb der Form-do-Routine passieren.

Ausstieg aus der Schleife, wenn "OK" oder "Cancel" geklickt wurde. Es werden bei jedem Schleifendurchlauf alle Buttons nach ihrem Status abgefragt - wie oben. Jetzt soll die Zahl, die in dem Button steht, in den Maskentext eingetragen werden. Es sind drei Möglichkeiten vorhanden, den einzutragenden Inhalt zu erfragen. Bei Buttons - und nur bei denen - steht, wie bekannt, im 12. bis 15. Byte der Objektstruktur die Adresse des Strings, der im Button sichtbar ist. Der kann also mit einem Verfahren, ähnlich wie Obtxt_get, ausgelesen werden. Oder es werden if..endif-Bedingungen gesetzt: Wenn Ret_ob=1, dann Alter=15, wenn Ret_ob=2, dann Alter=20 usw. Ich habe hier einfach aus

dem Index des Objekts den Wert herausgerechnet, indem ich den 5er Sprung der Zahlenreihe genutzt habe: Der erste Button hat den relativen Index 4, damit den absoluten Index T2_begin+4. Obstate_get gibt uns bekanntlich ebenfalls den absoluten Index zurück, so daß (N%-T2_begin%-4) eine Zahl zwischen 1 und Button-Anzahl zurückgibt.

Diese Zahl wird der Obtxt_set-Routine übergeben und die Box neu gezeichnet.

Falls die Box "gecancelt" wurde, muß das zuletzt angeklickte und selektierte Objekt normalisiert, das ursprünglich selektierte wieder invertiert und der ursprüngliche String wieder an den Maskentext übergeben werden.

Wenn Sie nach der Formularroutine für den ersten Objektbaum den Einsprung in diese Prozedur formulieren:

<Gosub Formular_bedienung>

Dann sind Sie mit dem zweiten Beispiel aus unserer Irrfahrt durch das AES-Labyrinth aus Bits, Bytes und Words fertig.

4.7 Imageaufbesserungen

Die Objektstruktur kann in dem Longword <ob_spec> (es wurde oben schon angedeutet) nicht nur auf die Sonderstruktur TEDINFO zeigen, sondern auch auf andere Sonderstrukturen, zum Beispiel BITBLK. Das Prinzip kann man sich genau wie bei TEDINFO vorstellen: Irgendwo im Speicher sind wieder einmal Bereiche reserviert, die Objektdaten enthalten, und zwar Daten über Bilder, die als Objekt (mit all seinen Möglichkeiten) ansprechbar sind. Sie werden sich fragen, was soll das, ich kann doch einfach ein Pixel-Muster auf den Bildschirm zeichnen und habe denselben Effekt. Aber weit gefehlt. Versuchen Sie einmal, solch ein Bild als Teil eines Formulars zu benutzen, es zu selektieren oder automatisch zeichnen und löschen zu lassen. Oder

überlegen Sie, welche Möglichkeiten für Datenbanken bestehen, wenn auf diese Art ein Bildobjekt integriert werden kann. Eine weitere feine Anwendung zeige ich Ihnen später noch.

4.7.1 Die BITBLK-Struktur

Also: Ob_spec zeigt auf die Anfangsadresse der BITBLK-Struktur. Diese besteht aus lediglich 14 Byte, aufgeteilt in:

1. Longword (Bi_pdata)

Anfangsadresse des Datenblocks, in dem das Pixel-Muster des Bildes steht.

2. Word (Bi_wb)

Breite des Bildes in Byte

3. Word (Bi_hl)

Höhe des Bildes in Pixeln

4. Word (Bi_x)

X-Koordinate des Bildes innerhalb des Objektbereichs.

5. Word (Bi_y)

Y-Koordinate des Bildes innerhalb des Objektbereichs. Das Objekt, in dem die Bilddaten zu integrieren sind, kann in seinen Grenzen größer sein als das eigentliche Bild. Mit dem 4. und 5. Word kann das Bild innerhalb dieses Rahmens verschoben werden.

6. Word (Bi_color)

Farbe des Musters, 0 für weiß, 1 für schwarz usw.

Weiter vorne im Buch wurde schon viel über Bildschirmorganisation gesagt: 640 Bits horizontal * 400 Bits vertikal sind 256000 Bits, die die Pixel-Informationen für den Schwarz-weiß-Bildschirm tragen können: 0 für weiß und 1 für schwarz. Eine Bildschirmzeile entspricht somit 640/8=80 Byte, der Bildschirm ist 80*400 Byte = 32000 Byte groß.

So einfach liegt der Bildschirm wirklich innerhalb des freien Speicherraumes. 32000 Byte hintereinander sind alle Pixel-Informationen eingetragen, angefangen bei Physbase, die durch <adr=Xbios(2)> zu erfragen ist.

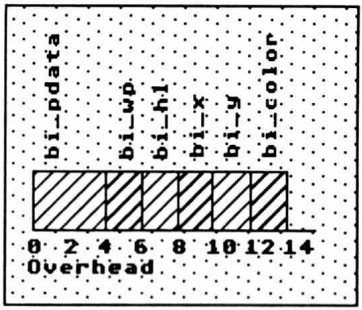


Abb. 17: BITBLK-Struktur

Und so einfach ist auch der Datenblock, der die Bilddaten beinhaltet, im Speicher zu organisieren: hintereinanderweg alle Bytes mit den entsprechenden Pixel-Informationen. Damit die Objc_draw Routine nun weiß, wann sie einen "Zeilenumbruch" zu machen hat, wird im BITBLK die Bildbreite in Byte angegeben, und damit sie weiß, wann Schluß ist, wie lang also der Bilddatenbereich innerhalb des RAM ist, wird die Pixel-Höhe des Bildes eingetragen. Dann ist Länge des Speicherbereichs in Byte = Breite%*Hoehe%. Und weil der 68000 Prozessor durch Byteberechnungen weit unterfordert wäre, weil er ja 16 Bit bzw. intern sogar 32 Bit gleichzeitig transportieren kann, arbeiten wir in 2-Byte-Einheiten, also Words.

So. Nun geben wir in einen reservierten Speicherbereich solcherart Bilddaten ein. Reserviert wird nach der schlichten, bekannten Methode:

```
Image$=Space$(Breite%*Hoehe%*2)
Image_adr%=Varptr(Image$)
```

In die einzelnen Byte dieses Strings poken wir jetzt nacheinander die Bilddaten ein. Entweder direkt:

```
Dpoke Image_adr%,&X1010001100011111
Dpoke Image_adr%+2,&X0000110011100011
```

usw., wobei jede 1 einem schwarzen und jede 0 einem weißen Punkt entspricht, ein Verfahren, das aber sehr mühsam und schreibaufwendig ist, oder wir lassen diese Binärzahlen in ihre Dezimalwerte umrechnen:

```
Dpoke Image_adr%,??????????
Dpoke Image_adr%+2,??????????
```

usw. Die einzelnen Word-Werte werden in DATA-Zeilen abgelegt und mit <Read> wieder herausgelesen.

Wie Sie diese Image-Daten produzieren können, zeigt Ihnen das nächste Kapitel.

Legen Sie jetzt mit Hilfe des RCS oder auch zu Fuß, wie im ersten Abschnitt beschrieben, die Objektstruktur an. Eine Box als Wurzel und ein Image als Unterobjekt sowie ein BUTTON mit Exit-Flag reichen. Ziehen Sie das Image soweit auf, daß Ihr Quellbildchen hineinpaßt, und machen Sie's nicht zu groß, die Handlichkeit des Programms schwindet mit der Größe des Bildes.

```
' ***** LIBRARY-ROUTINE *****
'
Procedure Imagedaten
  Local Breite%,Hoehe%,Dta%,I%
  Breite%=....                    !   in Words
  Hoehe%= ....
  Data 0,0,0,...
```

```
  Data ......
  Data 65535, .....
  '
  For I%=0 to Breite%*Hoehe%*2-1 Step 2
    Read Dta%
    Dpoke Image_adr%+i%,Dta%
  Next I%
  '
  Gosub Rsrc_gaddr(4,0,*Bitblk_adr)      ! Anfangsadresse des BITBLK
  Lpoke Bitblk_adr%,Image_adr%           ! Anfangsadresse der Bilddaten
  Dpoke Bitblk_adr+4,Breite%*2           ! Breite des Feldes in Byte
  Dpoke Bitblk_adr+6,Hoehe%              ! Höhe des Feldes in Pixeln
  Dpoke Bitblk_adr+8,0                   ! X-Koordinate
  Dpoke Bitblk_adr+10,0                  ! Y-Koordinate
  Dpoke Bitblk_adr+12,1                  ! Farbe
Return
```

Das war's auch schon. Starten Sie dieses 3. AES-Beispiel:

```
' Programm: IMAGE.BAS
'
Gosub Rsrc_load("IMAGE.RSC")
Gosub Rsrc_gaddr(0,0,*ad1)
Gosub Imagedaten
Gosub Objc_draw(ad1,0,2)
Gosub Form_do(ad1,0,*ret_ob%)
Gosub Rsrc_free
```

Ihr Bild erscheint auf dem Bildschirm.

Mit diesem Image-Objekt kann jetzt ebenso wie mit den übrigen Ihnen bekannten Objekten verfahren werden. Mit <Procedure Objc-change(...)> können Sie den Status_flag verändern, durch Manipulation des 8. und 9. Words der Objektstruktur ändern Sie die Koordinaten des IMAGE-Objekts innerhalb der Wurzelbox, Sie können auch, wenn Sie zwei Bilder einladen und zwei Speicherbereiche für die Bilddaten, aber eine BITBLK-Struktur reservieren, zwischen den beiden Bildern hin- und herschalten:

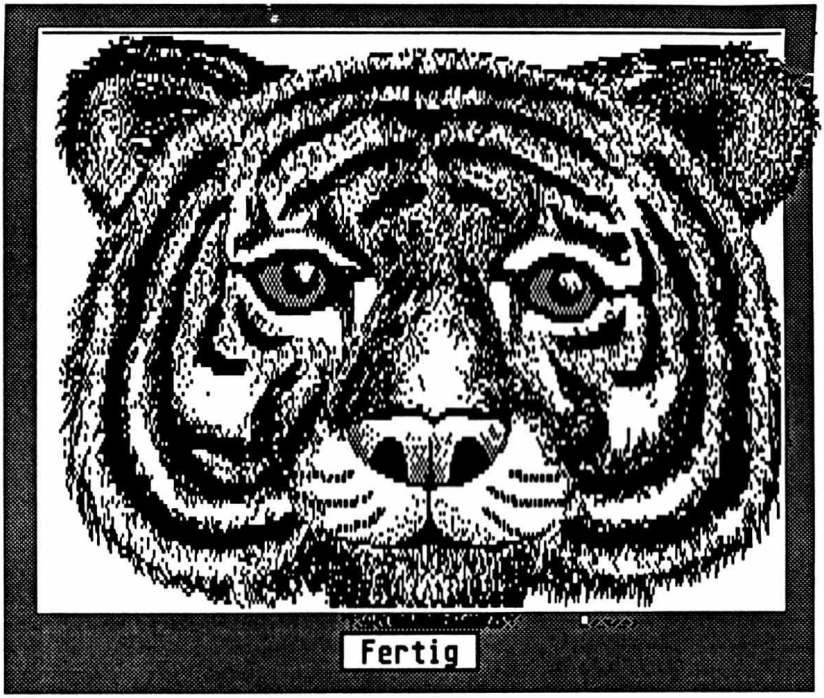


Abb. 18: Der Papiertiger: Als Objekt gebändigt

```
Repeat
  Lpoke Bitblk_adr,Bilddatenadresse_1
  Gosub Objc_draw(...)
  Lpoke Bitblk_adr,Bilddatenadresse_2
  Gosub Objc_draw(...)
Until Mousek=0
```

4.7.2 Der Image-Generator für die Bilddaten

Dieses einfache, aber recht wirkungsvolle Programm besteht aus zwei Prozeduren, einer, die das Bild, das in Objektdaten überführt werden soll, lädt und umwandelt, und einer, die es wieder auf die Diskette zurückspeichert.

Die <Procedure Image_find> lädt das File in den Bildschirmspeicher, und mit Hilfe des Mauszeigers läßt sich mittels zweier Schleifen ein Bereich aus diesem Bild auswählen, der in den String <Temp$> überführt wird. Dieser wird dann in die linke obere Ecke des Bildschirms geschrieben. Also nochmal: Es muß erreicht werden, daß auf möglichst einfache Art die Pixel-Information eines Bildschirmbereiches gelesen werden kann. Der Bildschirmbeginn bei <Xbios(2)> ist definiert, also packen wir das Bild dorthin, lesen Wort für Wort des Bildschirmspeichers aus und überführen diese Informationen in das zweidimensionale Array <Wort(x,y)>.

```
' Programm: IMG_EDIT.BAS
'
Procedure Image_find
  Dim Word(400,50)
  Fileselect "a:\*.*","",Name$
  If Name$<>""
    Bload Name$,Xbios(2)
    Do
      Mouse X%,Y%,K%
      Exit If K%=1
    Loop
    While K%=1
      Graphmode 3
      Mouse X1%,Y1%,K%
      Vsync
      Box X%,Y%,X1%,Y1%
      Vsync
      Box X%,Y%,X1%,Y1%
    Wend
    Get X%,Y%,X1%,Y1%,Temp$
    Cls
    Put 0,0,Temp$
    Breite%=Int(Abs((X1%-X%))/16)+1
    Hoehe%=Abs(Y1%-Y%)
    Adr=Xbios(2)
    For L%=1 To Hoehe%
      For M%=0 To Breite%-1
        Word(L%,M%)=Dpeek(Adr)
        Add Adr,2
      Next M%
      Add Adr,(40-Breite%)*2
    Next L%
```

```
    Endif
  Return
  '
  Procedure Image_save
    Fileselect "A:\*.icn","",Name$
    If Name$<>""
      Open "O",#1,Name$
      Print #1,"Breite%=";+Breite%
      Print #1,"Hoehe%=";+Hoehe%
      For N%=1 To Hoehe%
        Print #1,"data ";
        For M%=0 To Breite%-1
          Print #1,Str$(Word(N%,M%))+",";
        Next M%
        Relseek #1,-1
        Print #1
      Next N%
      Close #1
    Endif
  Return
  '
```

Mit der Zeile 'Relseek#1,-1'haben wir wieder einen kleinen Trick geschafft. Da nach jeder Zahl das Komma zum Trennen der einzelnen Datas geschrieben wurde, dieser aber beim letzten Eintrag einer Zeile falsch wäre, postieren wir den File-Pointer auf die Position des letzten Kommas und überschreiben dieses mit Print#1.

4.8 Neue Icons für unsere Programme

In Programmen wie ADIMENS oder dem Resource Construction Set finden Sie Icons, also kleine Desktop-Bildchen, die angeklickt werden und dadurch bestimmte Prozesse auslösen können. Auch diese Icons sind als Objekte innerhalb der Objektstruktur gelagert. Für sie zeigt das Longword <Ob_spec> auf die Adresse der ICONBLK-Struktur.

Icons unterscheiden sich von den oben erläuterten Images durch drei zusätzliche Features: Zum einen bestehen sie aus den Bilddaten und zusätzlich aus den dazugehörigen Maskendaten, d.h., daß sie invertiert auch wieder ein korrektes Bild ergeben, unab-

hängig davon, auf welchem Untergrund sie liegen. Sie werden also bedingungslos im Graphmode 1 dargestellt. Zum zweiten ist in der ICONBLK-Struktur ein Zeiger auf einen String enthalten, der z.B. bei den Desktop Icons für die Laufwerke die Laufwerksbezeichnung oder bei denen für die Disketten-Files deren Namen enthält, und zum dritten, ebenfalls innerhalb der Struktur eingebunden, gibt's einen Charakter, der weitere Spezifikationen angibt. Bei den Laufwerk-Icons kennzeichnet dieser Charakter den Laufwerkskennbuchstaben.

Am Anfang auch hier wieder die Struktur des Speicherbereichs, auf den ob_spec aus der Objektstruktur zeigt. Er ist diesmal 32 Byte lang:.

4.8.1 Die ICONBLK-Struktur

1.Longword (Ib_pmask)

Adresse des Speicherbereichs, in dem die Icon-Maskendaten abgelegt sind.

2. Longword (Ib_pdata)

Adresse des Speicherbereichs, in dem die Icon-Bilddaten abgelegt sind.

3. Longword (Ib_ptext)

Adresse des Speicherbereichs, in dem der zugehörige String abgelegt ist.

4. Word (Ib_char)

Der zugehörige Charakter (wie das "A" im Icon "Diskstation")

5. Word (Ib_xchar)

X-Koordinate dieses Charakters, relativ zur linken oberen Ecke des Icons, die in der Objektstruktur definiert ist.

6. Word (Ib_y_char)

Y-Koordinate dazu

7. Word (Ib_xicon)

X-Koordinate der linken oberen Ecke des Icons, auch relativ zur Hauptkoordinate in der Objektstruktur.

8. Word (Ib_yicon)

Y-Koordinate der linken oberen Ecke des Icons

9. Word (Ib_wicon)

Breite des Datenfeldes in Pixeln. Das muß eine Zahl sein, die ohne Rest durch 16 teilbar ist, also innerhalb von Word-Grenzen liegt.

10. Word (Ib_hicon)

Höhe des Datenfeldes in Pixeln.

11. Word (Ib_xtext)

X-Koordinate des Textes, relativ zur oberen linken Ecke des Icons aus der Objektstruktur.

12. Word (Ib_ytext)

Dazugehörige Y-Koordinate des Textes

13. Word (Ib_wtext)

Breite des Textes in Pixeln

14. Word (Ib_htext)

Höhe des Textes in Pixeln

Nach derselben Methode wie bei den Images weiter oben, nur jetzt mit insgesamt drei Datenfeldern, wird das Icon dargestellt. Wieder können mit Hilfe des Varptr Speicherbereiche reserviert und darein die Icon-Daten gepoked werden. Ich zeige hier zur Abwechslung eine zweite Methode, die allerdings ihre Grenzen bei zu großen Datenfeldern hat. Es werden hier Zahlen als Datenträger geliefert, die in einen String umzuwandeln sind. Genau dafür enthält GFA-BASIC den Befehl Mki$(), der eine Integerzahl in einen String verwandelt. Der Rest der <Procedure Icondefinition> ist jetzt für Sie nicht schwer zu lesen. Die in Data-Zeilen gefangenen Icon- und Maskendaten, hier abwechselnd Icon und Maske, werden in zwei Strings eingelesen, deren Anfangsadressen erfragt und wie gehabt der ICONBLK-Struktur übergeben. Ich habe hier der Verständlichkeit halber die Icon-Größe auf 48x48 Pixel festgesetzt und auch die Lage und Höhe der Charakter- und Stringdaten nicht variabel gemacht. Ihnen wird es sicherlich nicht schwerfallen, das mit den bisherigen Informationen zu verändern.

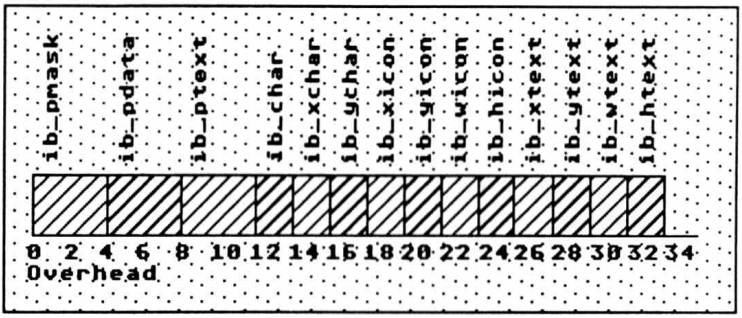


Abb. 19: ICONBLK-Struktur

```
Procedure Icondefinition(T.xt$,C.hr$)
  Local I%,Msk,Dta,
  ' Icon Datenblock
  Data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  Data ....
  Data *
```

```
  T.xt$=T.xt$+Chr$(0)
  '
  ' Icondaten umwandeln
  For I%=1 To 48*3
    Read A%
    Icond$=Icond$+Mki$(A%)
    Read B%
    Iconm$=Iconm$+Mki$(B%)
  Next I%
  '
  Dta=Varptr(Icond$)
  Msk=Varptr(Iconm$)
  Gosub Rsrc_gaddr(3,0,*Iconadr)
  '
  Lpoke Iconadr,Msk              ! Adresse des Datenblocks
  Lpoke Iconadr+4,Dta            ! Adresse des Maskenblocks
  Lpoke Iconadr+8,Varptr(Txt$)   ! Adresse des Strings
  Dpoke Iconadr+12,Asc(C.hr%)    ! Charakter
  Dpoke Iconadr+14,0             ! X-Koordinate des Charakters
  Dpoke Iconadr+16,0             ! Y-Koordinate des Charakters
  Dpoke Iconadr+18,0             ! X-Koordinate des Piktogramms
  Dpoke Iconadr+20,0             ! Y-Koordinate des Piktogramms
  Dpoke Iconadr+22,48            ! Breite% in Pixeln
  Dpoke Iconadr+24,48            ! Höhe% in Pixeln
  Dpoke Iconadr+26,0             ! X-Koordinate des Textes
  Dpoke Iconadr+28,40            ! Y-Koordinate des Textes
  Dpoke Iconadr+30,len(t.xt$)*8  ! Breite des Textes in Pixeln
  Dpoke Iconadr+32,8             ! Höhe des Textes in Pixeln
Return
```

Wenn Sie mit Hilfe des RCS ein Objekt ICON definiert haben, können Sie jetzt ein wunderschönes, selbstgebautes Icon auf dem Bildschirm haben. Es muß natürlich, um ansprechbar zu sein, mit dem SELECTABLE-Flag ausgestattet sein, und vergessen Sie nicht, daß irgendeins der Objekte das EXIT-Flag haben muß.

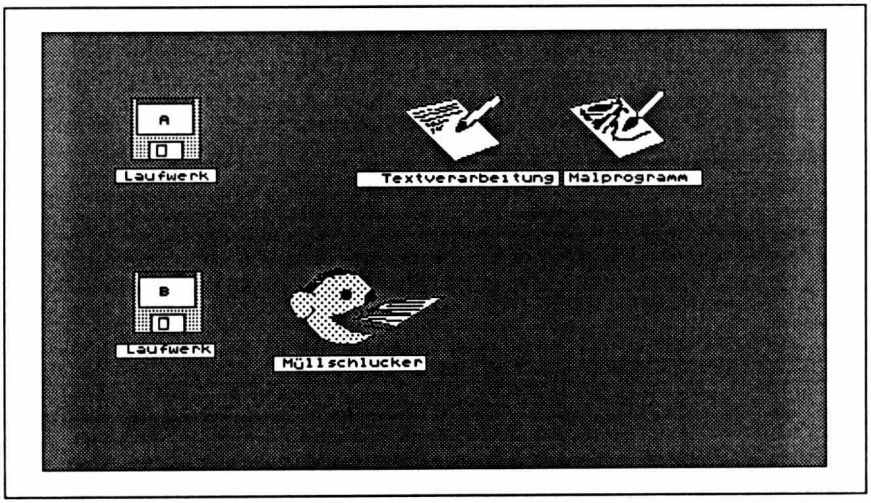


Abb. 20: Icons

Hier wieder die Formularroutine dazu. Da nichts Neues dabei ist, wird sie unkommentiert wiedergegeben:

```
Gosub Rsrc_load("icon.rsc")          ! RSC-File laden
Gosub Rsrc_gaddr(0,Tree1%,*Ad1)      ! Die Baumadresse
Gosub Icondefinition
Gosub Objc_draw(Ad1,0,3)             ! und Formular zeichnen
Gosub Form_do(Ad1,0,*Ret_ob%)        ! Die Formularroutine
Gosub Rsrc_free
```

4.8.2 Ein Icon-Editor für die Objektstruktur ICON

Noch haben Sie keine Daten, die Sie in die ICONBLK-Struktur einbinden könnten. Es muß also noch ein Editor her. Hier ist er, er besteht aus zwei Prozeduren zum Aufbau des Icons und zum Abspeichern. Drumherum sind noch einige Prozeduren gewebt, die den Bildschirm verschönern und die Bedienung erleichtern.

In Kapitel 3.1 haben Sie bereits einen ähnlichen Editor, "IMAGEDIT", kennengelernt. Bei diesem konnten Sie bei glei-

cher Maussteuerung IMAGES von 16*16 Bit bis 32*32 Bit erstellen. Je nach Aufgabe Ihres IMAGES konnten Sie Masken oder Datas zeichnen und als Binär- und Dezimaldatas abspeichern. In diesem Programm nun haben wir ein kleines Menü zur Verfügung und müssen uns nicht durch Alertboxen hangenln. Außerdem steht uns jetzt ein erweitertes Raster von 48*48 Bildpunkten zur Verfügung. Das Umschalten zur Masken- und Data-Erstellung ist möglich. Nun folgt wieder die Beschreibung der einzelnen Prozeduren.

Fangen wir mit der <Procedure Screen> an. Hier wird eine Überschrift gemalt, ein großes Karofeld ausgebaut, in dem das Icon und seine Maske gezeichnet wird. Eine kleine Box daneben zeigt das fertige Icon in Originalgröße, und ein einfacher Menübaum soll Verzweigungen zum Zeichnen der Maske, Abspeichern, Ausstieg usw. ermöglichen.

```
' Programm: ICN_EDIT.BAS
'
Procedure Screen
  Box 400,8,630,38
  Deftext 1,17,0,13
  Text 470,26,"ICONEDITOR"
  Deftext 1,0,0,4
  Text 450,35,"gebastelt von Udo Onnen"
  Deftext 1,0,0,6
  Text 457,115,"Icon"
  Text 465,125,"und"
  Text 450,135,"Maske"
  '
  Deftext 1,0,0,13
  Box 400,303,630,385
  Text 450,317,"Icondata"
  Line 400,319,630,319
  Text 450,333,"Iconmaske"
  Line 400,335,630,335
  Text 450,350,"Icon laden"
  Line 400,352,630,352
  Text 450,366,"Icon speichern"
  Line 400,368,630,368
  Text 450,382,"Ende"
```

```
    For N%=0 To 48*8 Step 8
      Line 8,8+N%,8+48*8,8+N%
      Line 8+N%,8,8+N%,8+48*8
    Next N%
    Box 500,100,550,150
  Return
```

Die <Procedure Zeichnen> ermöglicht die Eingabe von vergrößerten Pixeln in das Karofeld, eventuelles Löschen von Pixeln und malt das Ergebnis in Originalgröße in die Iconbox: Innerhalb einer Do..Loop-Schleife wird der Mauscursor abgefragt, ob er im Menüfeld oder im Zeichenfeld steht. Die Mauskoordinaten werden so verwandelt, daß jedes Kästchen eine von der linken oberen Ecke aufsteigende Zahlenfolge von 1 bis 48 erhält (sowohl waagerecht als auch senkrecht). Wenn innerhalb eines Kästchens die linke Maustaste gedrückt wird, wird dieses Kästchen schwarz gemalt und in ein zweidimensionales Array Mask%() bzw. Dta%() an der entsprechenden Zeilen- und Spaltenposition eine "1" geschrieben. Gleichzeitig wird an der entsprechenden Stelle in der Iconbox ein einzelner Punkt geplottet, bei der Maskeneingabe in einer einfachen, selbstdefinierenden Graphmode 3-Variante. Nach Anklicken der rechten Maustaste geschieht haargenau dasselbe, nur daß nun ein weißes Kästchen gefüllt und ein weißer Punkt geplottet wird.

```
  Procedure Zeichnen
    Do
      Mouse X%,Y%,K%
      If X%>400 And X%<630 And Y%>270 And Y%<386 And K%=1
        Gosub Menu
      Endif
      If X%>=8 And Y%>=8 And X%<8+48*8 And Y%<8+48*8
        X1%=(X% Div 8)*8
        Y1%=(Y% Div 8)*8
        If K%=1
          Deffill 1,1
          Pbox X1%,Y1%,X1%+8,Y1%+8
          If D!=True
            Dta(Y1%/8,X1%/8)=1
          Else
            Mask(Y1%/8,X1%/8)=1
            If Dta(Y1%/8,X1%/8)=1
              Color 0
```

```
            Else
              Color 1
            Endif
          Endif
          Plot 500+X1%/8,100+Y1%/8
          Color 1
        Else
          If K%=2
            Deffill 1,0
            Color 0
            Pbox X1%,Y1%,X1%+8,Y1%+8
            If D!=True
              Dta(Y1%/8,X1%/8)=0
            Else
              Mask(Y1%/8,X1%/8)=0
            Endif
            Plot 500+X1%/8,100+Y1%/8
            Color 1
          Endif
        Endif
      Endif
      Exit If K%=3
    Loop
  Return
```

Auf diese Weise haben wir in zwei Arrays Dta%() und Mask%() für jedes Iconpixel die Aussage schwarz oder weiß, gegliedert in die Piktogramm- und die Maskendaten, erhalten. In Computersprache: Wir haben die einzelnen Bitinformationen. Diese in die erforderlichen Worddaten zu verwandeln, sollte leicht sein und ist es auch. Es werden immer Pakete von 16 Bit geholt und nach der Binärformel x=2^0+2^1+2^2+....+2^n in eine Integerzahl umgerechnet. Diese Zahl wird dann sofort auf die Diskette gespeichert. Der Rest der Prozedur dient lediglich dazu, den Daten-File vom BASIC-Programm wieder einlesen zu können.

```
Procedure Icon_save
  Fileselect "*.*","",Name$
  If Name$<>""
    Open "O",#1,Name$
    C%=0
    Print #1,"data ";
    For N%=1 To 48
      For M%=1 To 16
```

```
        If Dta(N%,M%)=1
          Icn=Icn+2^(Abs(M%-16))
        Endif
        If Mask(N%,M%)=1
          Msk=Msk+2^(Abs(M%-16))
        Endif
      Next M%
      Print #1,Icn;",";Msk;",";
      Icn=0
      Msk=0
      For M%=17 To 32
        If Dta(N%,M%)=1
          Icn=Icn+2^(Abs(M%-32))
        Endif
        If Mask(N%,M%)=1
          Msk=Msk+2^(Abs(M%-32))
        Endif
      Next M%
      Print #1,Icn;",";Msk;",";
      Icn=0
      Msk=0
      For M%=33 To 48
        If Dta(N%,M%)=1
          Icn=Icn+2^(Abs(M%-48))
        Endif
        If Mask(N%,M%)=1
          Msk=Msk+2^(Abs(M%-48))
        Endif
      Next M%
      Print #1,Icn;",";Msk;
      Icn=0
      Msk=0
      Inc C%
      If C%<3
        Print #1,",";
      Else
        Print #1,
        Print #1,"data ";
        C%=0
      Endif
      Icn=0
    Next N%
    Print #1,"*"
    Close #1
  Endif
Return
```

Zwei weitere Prozeduren dienen der Erneuerung des Zeichenbereichs beim Wechseln von Piktogrammeingaben zu Maskeneingaben. Sie sind nicht weiter spannend, es werden die Arrays abgefragt und entsprechend weiße oder schwarze Pboxen gemalt. Zusätzlich wird natürlich in der Iconbox das bisher gezeichnete Piktogramm in Originalgröße dargestellt.

```
Procedure Icon_data
  D!=True
  Graphmode 1
  For N%=1 To 48
    For M%=1 To 48
      If Dta(N%,M%)=1
        Deffill 1,1
          Pbox M%*8,N%*8,M%*8+8,N%*8+8
        Color 1
        Plot 500+M%,100+N%
      Else
        Deffill 1,0
        Pbox M%*8,N%*8,M%*8+8,N%*8+8
        Color 0
        Plot 500+M%,100+N%
      Endif
    Next M%
  Next N%
  Gosub Zeichnen
Return
'
Procedure Icon_mask
  D!=False
  Graphmode 1
  For N%=1 To 48
    For M%=1 To 48
      If Mask(N%,M%)=1
        Deffill 1,1
        Pbox M%*8,N%*8,M%*8+8,N%*8+8
        Color 1
        Plot 500+M%,100+N%
      Else
        Deffill 1,0
        Pbox M%*8,N%*8,M%*8+8,N%*8+8
        Color 0
        Plot 500+M%,100+N%
      Endif
```

```
    Next M%
  Next N%
  Gosub Zeichnen
Return
```

Sinn macht dieser Aufwand natürlich erst dann, wenn durch eine Menüsteuerung auch gewährleistet ist, daß zwischen Daten- und Maskeneingabe hin- und hergeschaltet werden kann, daß bei Bedarf gesichert wird und daß ein ordnungsgemäßes Beenden des Programms möglich ist. Dafür also die Prozedur Menu, die die verschiedenen Prozeduren verwaltet. Wenn Sie sich die If..Endif Bedingungen für Icon_load und Icon_save ansehen, finden Sie übrigends eine einfache und problemlose Radio-Button-Routine, die Boxen wechselweise selektiert oder normalisiert. Es geht also auch ohne Objekte, nicht wahr?

```
Procedure Menu
  Graphmode 3
  If Y%>303 And Y%<319
    Gosub Icon_data
  Endif
  If Y%>319 And Y%<335
    Gosub Icon_mask
  Endif
  If Y%>335 And Y%<352
    Color 1
    Pbox 400,335,630,352
    Gosub Icon_load
    Color 0
    Pbox 400,335,630,352
  Endif
  If Y%>352 And Y%<368
    Color 1
    Pbox 400,352,630,368
    Gosub Icon_save
    Color 0
    Pbox 400,352,630,368
  Endif
  If Y%>368
    Color 1
    Pbox 400,368,630,385
    Alert 2,"Soll wirklich Schluß sein ?",2,"Ja|nein",Knopf%
    If Knopf%=1
      End
```

```
      Endif
      Color 0
      Pbox 400,368,630,385
    Endif
    Graphmode 1
  Return

  Procedure Icon_load
  '
  ' Diese Procedure sollten Sie nach den bisher beschriebenen Funk
  ' tionen selbst entwickeln können. Sie können sich dabei auf die
  ' Prozedur "Icon_save" beziehen, da Sie die Daten genauso
  ' wieder einladen können.
```

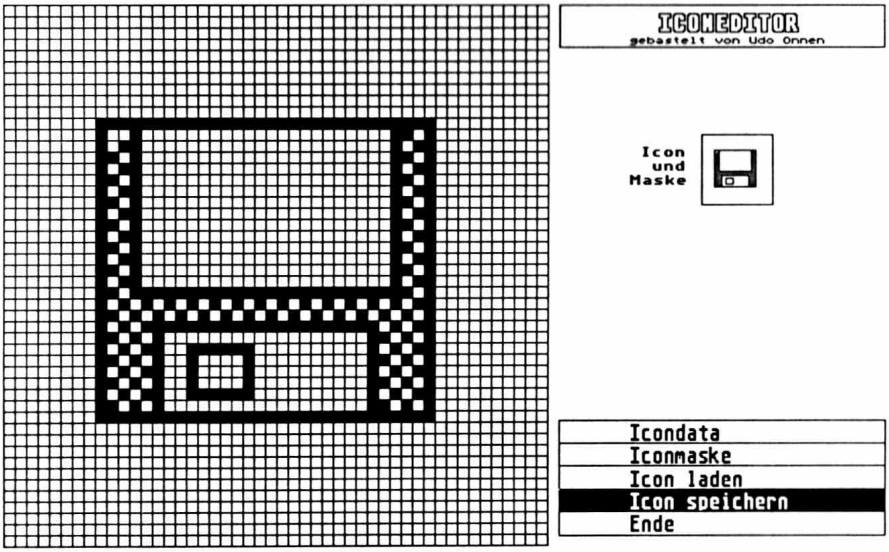


Abb. 21: Der Iconeditor

Die letzten vier Zeilen sind die ersten. Sie gehören an den Programmanfang, dimensionieren die Arrays und starten das Programm.

```
Dim Dta(48,48)
Dim Mask(48,48)
D!=True
'
Gosub Screen
Gosub Zeichnen
```

4.9 Die Krönung: Das private Desktop

Mit all der Erfahrung, die Sie jetzt haben, würde es Ihnen gelingen, für Ihre Programme neue Bedienungsoberflächen zu schaffen, die ästhetisch ansprechend und funktional durchdacht sind. Warum nicht ein Bild statt des tristen grauen Hintergrundes, warum nicht - wie z.B. in 1st-Word vorexerziert, die Funktionstastenbelegung und deren momentanen Status anzeigen lassen. Vieles, vieles mehr gäbe es zu sagen. Wenn nicht dieser so wunderschön durchdachte Hintergrund von Fenstern, Deskaccessories und anderen Objekten überschrieben werden würde. So geht's also nicht.

Aber wenn ich Ihnen jetzt verrate, daß in der "Wind_set"-Routine aus der Window_library ein Parameter so belegt werden kann, daß auf einmal eine Objektstruktur als Desktop generiert und verwaltet werden kann, so daß der Hintergrund nach Überlagerungen von Fenstern oder Acc's wieder erscheint, wie es ihm bestimmt ist, dann können Sie Ihr Ziel erreichen. Und wenn Sie jetzt noch feststellen werden, wie trivial das ganze Verfahren ist, werden Sie staunen.

Sie brauchen lediglich einen Resource-File zu generieren (ob mit Hilfe des RCS und unter Verlust von 32 KByte oder zu Fuß innerhalb des Programms, ist auch hier egal), diesen nach der bekannten Methode zu laden, seine Adresse zu erfragen, den zuständigen Objektbaum zu zeichnen und die Adresse der Wind_set-Routine zu übergeben.

Aber wie alles im Leben hat auch dies einen kleinen Haken, der allerdings im Resource Construction Set aufgehängt ist: bekanntlich kann dieses kein Objekt von ganzer Bildschirmbreite und -höhe erzeugen, weil es in einem Fenster mit seinen Rän-

dern arbeitet. Ein dort erzeugtes Objekt, das so groß wie der Bildschirm sein soll, muß innerhalb des laufenden Programms modifiziert werden.

Aber auch das Verfahren ist ohne Schwierigkeiten zu bewerkstelligen. Sie wissen, daß jeweils im 16., 18., 20. und 22. Byte der Objektstruktur die Objektkoordinaten liegen. Hier kann also mit einem einfachen Dpoke der Wert entsprechend geändert werden.

Das kleine Beispielprogramm, das ich Ihnen hier vorstelle, zeigt lediglich ein einfaches Bildchen aus Standard-Objekten zusammengesetzt und 10 Boxen am unteren Bildschirmrand, die z.B. die Funktionstasten darstellen können. Ein "Ostrowski-Fenster" wird mit einem Klick auf die linke Maustaste geöffnet und mit einem auf die rechte wieder geschlossen. Damit ist bewiesen, daß der Objektbaum neues Desktop geworden ist.

```
' Programm: DESKTOB.BAS
'
Gosub Rsc_init
Gosub Main
End
'
Procedure Rsc_init
  Gosub Rsrc_load("desktop.rsc")    ! Laden des RCS-Files
  Gosub Rsrc_gaddr(0,Tree1%,*Ad1)   ! Ermitteln der Adresse
  Dpoke Ad1+18,20                   ! Hier werden die Y-Koordi-
  Dpoke Ad1+20,640                  ! nate, die Breite und
  Dpoke Ad1+22,390                  ! Höhe der Wurzel geändert
  '
  For N%=1 To 19 Step 2             ! Auch die F-Tasten-Boxen
    Gosub Rsrc_gaddr(1,N%,*Ad)      ! müssen in zwei Koordi-
    Dpoke Ad+18,350                 ! natenwerten verändert
    Dpoke Ad+22,25                  ! werden.
  Next N%
  Dpoke Ad+20,55                    ! Anpassen der Box am rechten
  '                                 ! Bildschirmrand
  Gosub Objc_draw(Ad1,0,3)          ! der Baum wird gezeichnet
  Gosub Wind_set(Ad1)               ! Und Wind_set macht daraus
Return                              ! das Desktop
'
```

```
Procedure Main
  Do
    Exit If Mousek=3
    If Mousek=2 And Offen!=True     ! In den folgenden Zeilen
      Closew 3                      ! wird auf Mausklick das
      Offen!=False                  ! Fenster geöffnet und
    Endif                           ! wieder geschlossen.
    If Mousek=2 And Offen!=False
      Openw 3
      Clearw 3
      Offen!=True
    Endif
    '
    Gosub Objc_find(Ad1,*Ob%)       ! Hier wird's wieder spannend:
    If Mousek=1 And Ob%<>0          ! Objc_find gibt den Index des
      Gosub Obstate_get(Ob%,*State%) !Objekts zurück, über dem
      If State%=32                  ! der Mauszeiger steht. Wenn
        Gosub Objc_change(Ad1,Ob%,1,0,0,640,400,1) ! dieses
      Endif                         ! Objekt selektiert ist, wirds
      If State%=1                   ! normalisiert, ansonsten selek-
        Gosub Objc_change(Ad1,Ob%,32,0,0,640,400,1)  !tiert. Die
      Endif                         ! Routinen Obstate_get und
      Pause 10                      ! Objc_change wurden früher
    Endif                           ! schon vorgestellt.
  Loop
  Closew 3
  Gosub Rsrc_free
Return
'
******* Library-Routinen ****************
Procedure Wind_set(A.)              ! Hier also der Wind_set.
  Dpoke Gintin,0                    ! Gintin erhält das Handle
  Dpoke Gintin+2,14                 ! des Desktops (0), Gintin+2
  Lpoke Gintin+4,A.                 ! braucht den Flag 14, der
  Dpoke Gintin+8,0                  ! das Desktop kreiert und
  Gemsys 105                        ! in Gintin+4 wird die Baum-
Return                              ! adresse übergeben.
'
Procedure Objc_find(A.,O.%)         ! Objc_find funktioniert
  Gemsys 79                         ! nicht mit dem Mouse x,y,k-
  Mx%=Dpeek(Gintout+2)              ! Befehl aus BASIC, deshalb
  My%=Dpeek(Gintout+4)              ! eine AES-Routine. Diese
  Dpoke Gintin,0                    ! Mauskoordinaten werden
  Dpoke Gintin+2,5                  ! übergeben, heraus kommt der
```

```
  Lpoke Addrin,A.                    ! Index des Objekts, auf dem
  Dpoke Gintin+4,Mx%                 ! die Maus steht.
  Dpoke Gintin+6,My%
  Gemsys 43
  *O.%=Dpeek(Gintout)
Return
```

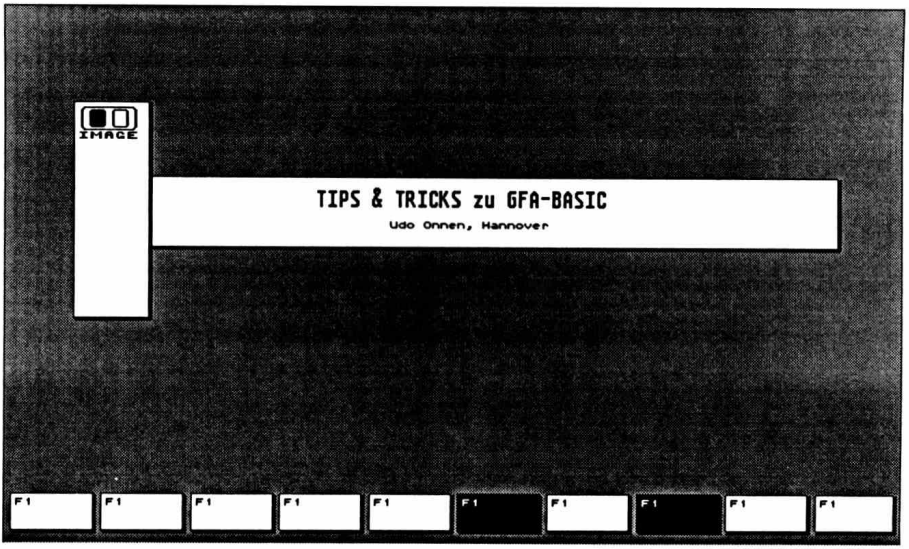


Abb. 22: Das eigene Desktop

Versuchen Sie einmal, Icons, Images und sonstiges Interessante in ein Desktop zu legen. Der Kreativität sind kaum Grenzen gesetzt.

4.10 Die Bibliothek des Grafen

Unter den diversen Einzelbibliotheken, die in AES zusammengefaßt sind, befindet sich eine adlige: Die Graf-Library. Wie's bei feudalen Dingen manchmal so ist, befindet sich in dieser Bibliothek mehr Schein als Sein, soll heißen, viel unnötige Wirkungen, wenig Effektivität. Insgesamt 10 Routinen finden wir hier:

GRAF_GROWBOX zeichnet ein sich automatisch stetig vergrößerndes Rechteck.

GRAF_SHRINKBOX zeichnet ein sich automatisch stetig verkleinerndes Rechteck.

GRAF_MOVEBOX zeichnet ein sich automatisch stetig von einer Position zur anderen bewegendes Objekt.

Alle drei Prozeduren sind meines Erachtens sinnlos, der Effekt ist zu schnell, als daß er Wirkung zeigt. Erlauben Sie mir, Sie bei Bedarf auf andere Literatur zu verweisen.

GRAF_RUBBERBOX Zeichnet "Hilfslinien" beim Vergrößern oder Verkleinern eines vorgegebenen Rahmens.

GRAF_DRAGBOX Zeichnet "Hilfslinien" beim Bewegen eines vorgegebenen Rahmens. Hält den Mauszeiger innerhalb dieses Rahmens fest.

Diese zwei Prozeduren finde ich ganz witzig. Deshalb hier ihre Erläuterung: Übergeben wird bei <Graf_rubberbox> der momentane x%- und der momentane y%-Wert, beides Koordinaten, die beim Vergrößern und Verkleinern unverändert bleiben. Zurückgegeben wird die neue Breite und die neue Höhe des Rahmens. Während des Arbeitsvorgangs werden gestrichelte Rahmenhilfslinien gezeichnet, die nach dem Loslassen des Mausknopfes wieder verschwinden.

<Graf_dragbox> will alle vier Rahmenkoordinaten haben. Da es sich um eine Funktion handelt, die die Bewegung auf Grenzkoordinaten beschränken kann, können diese noch eingegeben werden, zurück kommen die neuen Höhen- und Breitenwerte. Der die

Verschiebung begrenzende Rahmen wird wohl meistens der Bildschirmbereich sein, dann müssen also die Werte 0,0,640,400 übergeben werden.

```
Procedure Graf_rubberbox(Gx%,Gy%,Grw%,Grh%)
  Dpoke Gintin,Gx%
  Dpoke Gintin+2,Gy%
  Dpoke Gintin+4,5
  Dpoke Gintin+6,5
  Gemsys 70
  *Grw%=Dpeek(Gintout+2)
  *Grh%=Dpeek(Gintout+4)
Return
'
Procedure Graf_dragbox(Gx%,Gy%,Gw%,Gh%,Rx%,Ry%,Rw%,Rh%,Grx%,Gry%)
  Dpoke Gintin,Gw%
  Dpoke Gintin+2,Gh%
  Dpoke Gintin+4,Gx%
  Dpoke Gintin+6,Gy%
  Dpoke Gintin+8,Rx%
  Dpoke Gintin+10,Ry%
  Dpoke Gintin+12,Rw%
  Dpoke Gintin+14,Rh%
  Gemsys 71
  *Grx%=Dpeek(Gintout+2)
  *Gry%=Dpeek(Gintout+4)
Return
```

GRAF_WATCHBOX prüft, ob sich der Mauszeiger innerhalb eines Rahmens befindet. Diese Funktion ist ebensogut mit <On Menu Ibox/Obox> zu bewerkstelligen.

GRAF_MOUSE Verändert den Mauszeiger. Auch hierfür hat GFA-BASIC einen eigenen Befehl.

GRAF_MKSTATE Gibt Mauskoordinaten, Mausknopfstatus und Status der Tastatursondertasten zurück. Diese Funktion haben Sie bereits in der <Procedure Objc_find> kennengelernt.

GRAF_HANDLE Gibt das VDI-Handle zurück. Wird in BASIC nicht benötigt.

Im folgenden stelle ich Ihnen ein kleines Beispiel vor, das mit Hilfe von <Graf_rubberbox> und <Graf_dragbox> eine Box verschiebt bzw. seine Größe verändert:

```
' Programm: GRAF_LIB.BAS
'
X%=100
Y%=100
W%=100
H%=100
Box X%,Y%,X%+W%,Y%+H%
Graphmode 3
Do
  While Mousek=1
    Xr%=X%
    Yr%=Y%
    Wr%=W%
    Hr%=H%
    Mouse Mx%,My%,K%
    If Mx%>X%+W%-3 And Mx%<X%+W%+3 And My%>Y%+H%-3 And My%<Y%+H%+3
      Gosub Graf_rubberbox(X%,Y%,*W%,*H%)
    Else
      If Mx%>X% And Mx%<X%+W%-3 And My%>Y% And My%<Y%+H%-3
        Gosub Graf_dragbox(X%,Y%,W%,H%,0,0,640,400,*X%,*Y%)
      Endif
    Endif
    Box Xr%,Yr%,Xr%+Wr%,Yr%+Hr%
    Box X%,Y%,X%+W%,Y%+H%
  Wend
Loop
```

Ein anderes Beispiel findet sich im Resources-Bausatz am Ende dieses Buchabschnitts. Dort werden die Prozeduren in Fenstern verwandt, was wegen des um 19 Pixel verkleinerten Bildschirmarbeitsbereichs etwas aufwendiger zu bewerkstelligen ist.

4.11 Menü mit mehreren Gängen

GFA-BASIC hat nach meiner Meinung das Menü-Handling des AES vorzüglich übersetzt. Mit wirklich einfachen Befehlsfolgen ist alles das, was GEM-Menüs auszeichnet, erreichbar. Da braucht's wohl keine Tips & Tricks.

Eins aber möchte ich Ihnen nicht vorenthalten: nämlich die Chance, in das Menü auch etwas anderes als Text zu legen, z.B. Bildchen, ja sogar Icons. Einige Zeichenprogramme führen die Vorteile schon vor, indem etwa die verschiedenen Füllmuster im Menü ansprechbar sind. Gehen wir noch einen Schritt weiter und generieren ein Menü folgenden Inhalts:

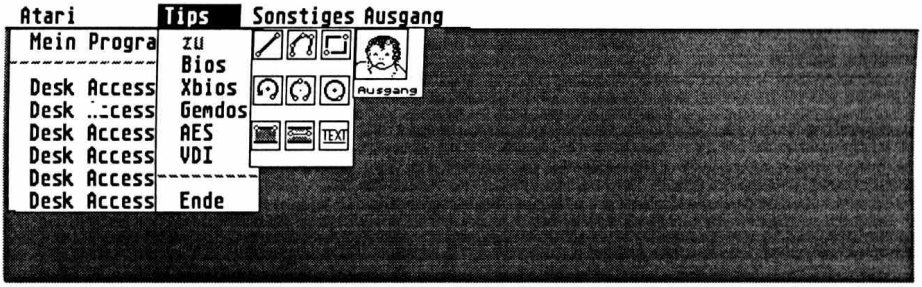


Abb. 23: Das Menü

Staunen Sie nicht zu sehr, das ist gar nicht schwierig. Nur müssen wir hier auch mal wieder etwas tiefer ins AES einsteigen.

Dieses Menü ist nicht mehr unter BASIC zu erstellen. Das Resource Construction Set muß her.

Ziehen Sie erst einmal einen Menü-Baum herunter. Zusätzlich (obwohl Sie den gar nicht haben wollen) einen Free-Baum. In diesen bauen Sie jetzt alles das herein, was Ihnen wichtig, nützlich oder sinnvoll erscheint. Zum Beispiel ein Image.

Wenn Sie dieses auf das Clipboard legen, zum Menü-Baum umschalten und dann das entsprechende Objekt in das Menü einbinden, haben Sie den Effekt erreicht. Werfen Sie den Free_baum in den Papierkorb, fertig.

Der Rest ist reinstes AES:

Das, was als Menü ins Programm eingebunden werden soll, ist nun ein Objektbaum, der mit <rsrc_load> in den Speicher gebracht werden muß und dessen Adresse mit <Rsrc_load> erfragt werden muß. Aber: Zeichen des Objekts und Statusveränderungen geschehen mit speziellen AES-Routinen zum Menü: der MENU-Library.

Ich gehe im folgenden davon aus, daß die Menü-Baum-Adresse grundsätzlich den Namen "Menu_adr%" erhält und als globale Variable definiert ist.

<Procedure Menu_bar()> zeichnet den Menü-Baum auf den Bildschirm. Im Parameter "F.%" übergeben Sie eine 1, wenn das Menü sichtbar, eine 0, wenn es unsichtbar sein soll.

```
Procedure Menu_bar(F.%)
  Dpoke Gintin,F.%
  Lpoke Addrin,Menu_adr%
  Gemsys 30
Return
```

Es sind vier weitere Routinen vorhanden, die das Menü beeinflussen können:

```
Procedure Menu_ienable(O.%,F.%)
  Dpoke Gintin,O.%
  Dpoke Gintin+2,F.%
  Lpoke Addrin,Menu_adr%
  Gemsys 32
Return
```

Hier wird ein Menüeintrag aktiviert bzw. deaktiviert, d.h. mit halber Helligkeit bzw. normal geschrieben. Im Parameter O.% ist der Objektindex zu übergeben, das Flag F.% signalisiert bei 1 den aktiven und bei 0 den inaktiven Eintrag.

Sie sehen, auch hier wird mit Objektindizes gearbeitet, anders als bei Herrn Ostrowski, der mit Menüeintragsindizes arbeitet. Es ist also angebracht, die Einträge im RCS mit Namen zu versehen und mit dem Header-Converter umzuwandeln.

```
Procedure Menu_tnormal(0.%)
  Dpoke Gintin,0.%
  Dpoke Gintin+2,1
  Lpoke Addrin,Menu_adr%
  Gemsys 33
Return
```

Diese Prozedur normalisiert invertierte Menütitel. Auch hier wird lediglich der Objektindex des Titels übergeben. Falls Sie es einmal brauchen sollten (ich wüßte nicht, wozu), können Sie in Gintin+2 eine 0 einpoken, dann haben Sie einen Titel invertiert.

```
Procedure Menu_text(0.%,T.ext$)
  Dpoke Gintin,0.%
  Lpoke Addrin,Menu_adr%
  T.ext$=T.ext$+Chr$(0)
  Lpoke Addrin+4,Varptr(T.ext$)
  Gemsys 34
Return
```

Diese letzte Menü-Prozedur verändert den Textinhalt eines Eintrags oder Titels. Auch sie will den Objektindex und zusätzlich den neuen Eintrag übergeben haben. Dies funktioniert besser als von Herrn Ostrowski vorgesehen, meine ich, denn in GFA-BASIC muß immer erst das alte Menü ausgeschaltet, der neue Eintrag in das Array eingegeben und dieses neue Menü wieder eingeschaltet werden. Hier reicht der Sprung zur <Procedure Menu_text>, um den neuen Eintrag im Menü sehen zu können.

Versuchen wir nun ein Beispiel. Die Konstantennamen aus dem konvertierten "*.H"-File stehen in der <Procedure Rsc_data>. Das Laden des RSC-Files und Erfragen der Menübaum-Adresse erfolgt bei <Procedure Rsc_init>. Das ist alles bekannt.

```
Procedure Rsc_data
  Let Menu_tree%=0  !/* TREE */
  Tips%=4
  T1%=19  !/* OBJECT in TREE #0 */
  T2%=20  !/* OBJECT in TREE #0 */
  T3%=21  !/* OBJECT in TREE #0 */
  T4%=22  !/* OBJECT in TREE #0 */
  T5%=23  !/* OBJECT in TREE #0 */
  Q.uit%=25  !/* OBJECT in TREE #0 */
  Myprog%=9  !/* OBJECT in TREE #0 */
  Ausgang%=37
Return
'
Procedure Rsc_init(N.ame$)
  Gosub Rsrc_load(N.ame$)
  Gosub Rsrc_gaddr(0,Menu_tree%,*Menu_adr%)
  On Menu  Gosub Do_menu
Return
```

Unser eigentliches Programm besteht dann nur aus den Sprüngen in die beiden Initiierungsprozeduren, dem Zeichnen des Menüs und einer Do..Loop-Schleife mit dem <On Menu>-Befehl.

```
Gosub Rsc_data
Gosub Rsc_init("Menu.rsc")
Gosub Menu_bar
Do
  Exit If E.flag!
  On Menu
Loop
Gosub Rsrc_free
```

Erst in der Arbeitsroutine <Procedure Do_menu> wird es interessant: Wir erkennen nämlich, daß mit den herkömmlichen Daten aus dem Ostrowskischen Menu_Array wenig anzufangen ist. Nicht mehr in Menü(0) steht der Index des gewählten Menüpunktes, sondern in Menü(5). Und der Titel des Menüs, der wegen der Übergabe an <Menu_tnormal> jetzt wichtig wird, ist in Menü(4) zu finden.

Also: Falls der angewählte Menüeintrag den Index "Myprog%" hat, wird der Eintrag geändert. Beim Index "Ausgang%" wird das Exitflag auf True gesetzt, die Do..Loop-Schleife wird abgebro-

chen. Falls der zum angewählten Eintrag gehörige Menütitel den Index "Tips%" hat, wird der angeklickte Eintrag inaktiviert, es sei denn, er hat den Index "Q.uit%". Dann werden alle unter diesem Titel befindlichen Einträge wieder aktiviert. Mit dem Normalisieren des invertierten Menütitels beenden wir unser Menü. Ich hoffe, es hat Ihnen geschmeckt.

```
Procedure Do_menu
  If Menu(5)=Myprog%
    Gosub Menu_text(Myprog%," Dein Programm ")
  Endif
  If Menu(5)=Ausgang%
    E.flag!=True
  Endif
  If Menu(4)=Tips%
    If Menu(5)=Q.uit%
      For N%=T1% To T5%
        Gosub Menu_ienable(N%,1)
      Next N%
    Else
      Gosub Menu_ienable(Menu(5),0)
    Endif
  Endif
  Gosub Menu_tnormal(Menu(4),1)
Return
```

4.12 Fensterln

Lieber Frank Ostrowski, so genial Ihr BASIC ist, so unverständlich sind Ihre Window-Routinen. So, wie das GFA-BASIC die Fenster vorgibt, sind sie sicherlich nur in Einzelfällen verwendbar.

Aber Gemach, liebe Programmierer. Die Windtab-Tabelle und das GEM ermöglichen uns trotz alledem einen professionellen Umgang mit bis zu 4 eigenständigen Fenstern, mit Tricks sogar bis zu 7 bzw. 8 davon.

Ein einzelnes Fenster so zu öffnen, wie es gebraucht wird, ist schlicht. In das Windtab-Array werden an die entsprechenden

Stellen die Koordination und die Randelementebelegung eingepoked und erst danach, wie im Handbuch beschrieben, die Openw-Routinen benutzt:

```
' Programm: FENSTER1.BAS
'
*************Library_routine************
Procedure Single_window(T.itle$,I.nfo$,R.and%,W.x%,W.y%,W.w%,W.h%)
  Titlew 1,T.itle$
  Infow 1,I.nfo$
  Dpoke Windtab+2,R.and%
  Dpoke Windtab+4,W.x%
  Dpoke Windtab+6,W.y%
  Dpoke Windtab+8,W.w%
  Dpoke Windtab+10,W.h%
  Openw 1
  Clearw 1
Return
```

Wenn Sie jetzt dieses Fenster mit <Closew 1> wieder schließen, kann es später jederzeit einfach mit <Openw 1> erneut geöffnet werden. Windtab hat die eingepokten Werte nicht vergessen.

Die einzelnen Parameter sind im Handbuch beschrieben bzw. für sich verständlich. Lediglich die Randkomponenten bedürfen einer Erläuterung.

Ein GEM-Fenster besteht aus 12 Randkomponenten und dem Arbeitsfeld. Die Randkomponenten sind in einem Word enthalten, dessen erste 12 Bits über die Aktivierung oder Deaktivierung Auskunft geben: Ein gesetztes Bit heißt: Komponente ist vorhanden, ein leeres Bit (0) heißt: nicht vorhanden. Sie können also, wenn Sie einmal keine Lust haben, den Parameter R.and% der obigen Prozedur auszurechnen, für die Aktivierung aller Randkomponenten auch eingeben:

```
R.and%=&X111111111111
```

Das vorweg. Die Bedeutung der einzelnen Bits und deren umgerechnete Integerwerte zeigen die folgende Tabelle und die Abbildung. Durch Addition der einzelnen Zahlen können Sie die Komponenten kombinieren.

1	NAME
2	CLOSER
4	FULLER
8	MOVER
16	INFO
32	SIZER
64	ARROW_UP
128	ARROW_DN
256	VSLIDE
512	ARROW_LF
1024	ARROW_RT
2048	HSLIDE

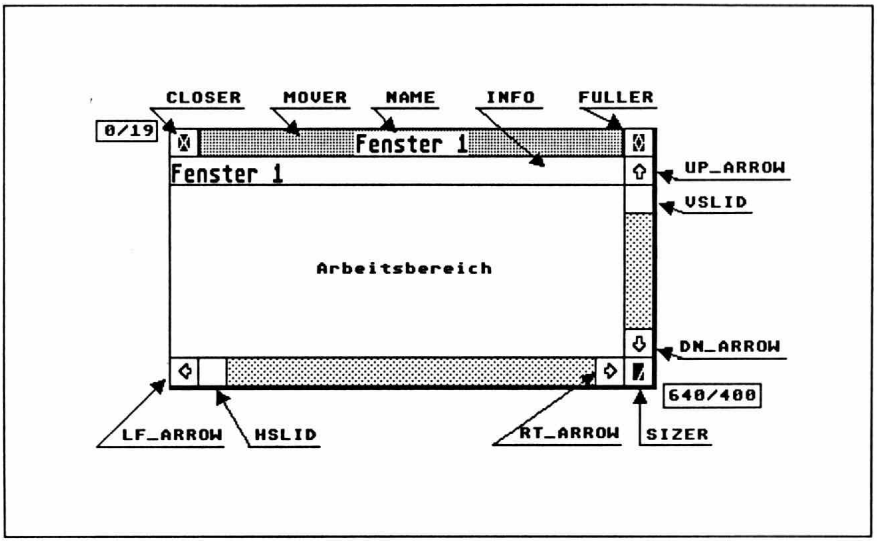


Abb. 24: Fensterelemente

Auch das Öffnen von mehreren Fenstern (bis zu 4 Stück) ist dank der Windtab-Tabelle unproblematisch. GEM verwaltet für jedes offene Fenster ein "handle", also eine Kennzahl. Diese ist für das Desktop, das (wie im Kapitel über das eigene Desktop erläutert) im Prinzip ebenfalls als Fenster verwaltet werden kann), die 0. Die weiteren Fenster erhalten Handles von 1 bis 4.

Mit Hilfe dieses Handle lassen sich später die Fenstermanipulationen exakt auf jeweils ein Fenster beziehen. In der Windtab-Tabelle sind 5 Bereiche vordefiniert, in denen die entsprechenden Fensterparameter aufbewahrt werden:

Windtab+00 bis Windtab+10	Handle=1
Windtab+12 bis Windtab+22	Handle=2
Windtab+24 bis Windtab+34	Handle=3
Windtab+36 bis Windtab+46	Handle=4
Windtab+52 bis Windtab+58	Handle=0

4.12.1 Sesam öffne dich

Damit sind die Grundlagen gelegt, um die multifunktionale Prozedur für das Öffnen mehrerer Fenster zu generieren. Es werden dieselben Parameter wie oben übergeben. Dann allerdings wird die Windtab-Tabelle daraufhin abgefragt, welches Handle noch nicht besetzt ist, also eine 0 als Eintrag hat. Die Fensterparameter werden nun nicht mehr bedingungslos in die ersten Felder der Windtab-Tabelle gepoked, diese werden mit dem Offset (Wi_handle%*12) errechnet. So können Sie sicher sein, daß immer ein freies Handle besetzt wurde. Wenn die Windtab-Tabelle voll ist, wird eine Alert-Meldung ausgegeben. Das Wi_handle% wird als globale Variable zurückgegeben.

Eine Zeile in der untenstehenden Prozedur bedarf einer genaueren Erläuterung. Wenn Sie später so weit sind, ein derzeit nicht aktives Fenster anklicken zu können und damit zwischen Fenstern hin- und herschalten zu können, brauchen Sie die Reihenfolge, in der die Fenster hintereinander aktiviert wurden. Die Fachleute reden immer von dem "Slot", dem Schlitz, in die die Manipulationsroutinen eingebaut werden müssen. Ich baue die Kennung für die Fensterreihenfolge in einen String <Slot$> ein, indem die einzelnen Handles hintereinander eingeschrieben werden. Jeweils die letzte in <Slot$> stehende Zahl kennzeichnet das aktuelle Window_handling.

```
*************Library_routine*************
Procedure Open_window(Wtit$,Winf$,Wat%,Wx%,Wy%,Ww%,Wh%)
  Local C%
  C%=0
  Wi_handle%=1
  Repeat
    Exit If Dpeek(Windtab+C%)=0
    Add C%,12
    Inc Wi_handle%
  Until Wi_handle%=5
  '
  If Wi_handle%<5
    Titlew Wi_handle%,Wtit$
    Infow Wi_handle%,Winf$
    Dpoke Windtab+Wi_handle%*12-10,Wat%
    Dpoke Windtab+Wi_handle%*12-8,Wx%
    Dpoke Windtab+Wi_handle%*12-6,Wy%
    Dpoke Windtab+Wi_handle%*12-4,Ww%
    Dpoke Windtab+Wi_handle%*12-2,Wh%
    Openw Wi_handle%
    Clearw Wi_handle%
    Slot$=Slot$+Str$(Wi_handle%)
  Else
    Alert 1,"Kein Fenster mehr verfügbar!",1,"  OK  ",Ret%
    Wi_handle%=4
  Endif
Return
```

4.12.2 Fensterbewegungen

Jetzt wird's spannend! Fenster wollen bewegt, verschoben, vergrößert und verkleinert werden. Mal ein nicht aktives Fenster in den Vordergrund bringen? Volle Größe und wieder auf das vorherige Maß zurückbringen? Kein Problem mit GEM. Alle diese Funktionen werden automatisch überwacht und ausgeführt. Sie selbst brauchen sich nur um das zu kümmern, was innerhalb des Arbeitsbereichs passiert.

Der Event-Handler bringt eine Message, die sich auf die Fensterbedienung bezieht: bei Ostrowski Menü(1). Folgende Werte für Menü(1) sind von Bedeutung:

20	*WM_REDRAW*	Fensterbereiche müssen neu gezeichnet werden
21	*WM_TOPPED*	ein nicht aktives Fenster soll aktiviert werden
22	*WM_CLOSED*	das Closer-Element wurde angeklickt
23	*WM_FULLED*	das Fuller-Element wurde angeklickt
24	*WM_ARROWED*	einer der Pfeile oder ein Scrollbalken wurde angewählt.
25	*WM_HSLID*	der horizontale Schieber wurde angewählt
26	*WM_VSLID*	der vertikale Schieber wurde bewegt
27	*WM_SIZED*	das Größenfeld wurde gewählt
28	*WM_MOVED*	das Kopffeld des Fensters wurde gewählt

Bei diesen Meldungen werden erläuternde Zusätze in Menü(4) bis Menü(8) abgegeben. Für alle Menü(1)-Werte gleich ist, daß Menü(4) immer das Wi_handle% trägt, auf das sich die Meldung bezieht. Überall da, wo die Fensterkoordinaten benötigt werden, also bei WM_REDRAW, WM_SIZED und WM_MOVED, werden in Menü(5) bis Menü(8) die X- und Y-Koordinate und die Breite und Höhe des Fensters ausgegeben. WM_HSLID und WM_VSLID tragen in Menü(5) die Slider-Position (darauf komme ich noch zu sprechen), und WM_ARROWED spezifiziert in Menü(5) näher, welcher Pfeil nun gerade angeklickt worden ist.

Dabei ist vielleicht interessant, daß GEM selbst nicht die entsprechende, zu der Meldung passende Routine ausführt, sondern nur diese Meldung bringt. Was Sie damit machen, ist Ihre Sache. Sie könnten auch den Closer- und Fuller-Mechanismus umdrehen oder ganz andere Bedienungen auf die Randelemente legen. Wir bleiben aber beim Bekannten.

Die <Procedure Window_handling> wird durch <On Menu Message Gosub Window_handling> aufgerufen. Mit der AES-Funktion <Wind_update> kann sichergestellt werden, daß das GEM die automatischen Routinen störungsfrei ausführen kann. Solange <Wind_update> durch Übergabe des Parameters 1 eingeschaltet ist, kann weder der Mauscursor bewegt, noch die Menüleiste bedient werden. Auch Accessories sind nicht ausführbar. Der Parameter 0 schaltet AES wieder ein.

Die Redraw-Routinen werden im nächsten Kapitel angesprochen. Aus dieser Prozedur werden nur das Handle des Fensters und die Koordinaten des Bereichs, der neu gezeichnet werden muß, weitergegeben.

Wenn ein nicht aktives Fenster mit der Maus angeklickt wurde, gibt AES eine Meldung aus. Dummerweise steht in Menü(4) aber nicht das Handle des neuen, sondern das des alten Fensters. Das brauchen wir aber gar nicht mehr. Also kommt <Wind_find> zum Tragen. Ähnlich wie bei <Objc_find>, das Sie schon kennengelernt haben, übergeben wir die Mauskoordinaten aus der AES-Funktion <Graf_mouse> und erhalten das Handle des Fensters unter dem Mauszeiger zurück. Dieses Fenster wird geöffnet, und jetzt kommt unser Slot in Aktion. Es wird nachgeschaut, an welcher Position dieses neue Fenster bisher geführt wurde, an dieser Position wird es im Slot gelöscht und an die letzte Stelle geschrieben. Damit ist sichergestellt, daß die vorherige Reihenfolge der Fenster gewahrt bleibt.

Full_window ist eine Schalterfunktion. Beim erstmaligen Anklicken wird das Fenster auf seine maximale Größe geöffnet, beim nächsten Anklicken soll es wieder an der vorherigen Position und in der vorherigen Größe erscheinen. Wir müssen also die Koordinatenwerte retten können und in einem Flag den derzeitigen Status kennzeichnen. Dienlich sind uns das Array Fullw%(4,4) und das Flag Fullw!. Zunächst werden die momentanen Fensterkoordinaten in das Array geschrieben. Diese erhalten wir aus der Windtab-Tabelle. Anschließend werden die Werte für die Fenstermaximalgröße in die Windtab-Tabelle eingepoked. Diese erhalten wir auch wieder aus dieser Tabelle: Es sind die Koordinatenwerte des Desktops, also des Handles 0. Wenn das Fenster wieder auf die alte Größe schrumpfen soll, werden einfach die Werte aus dem "Fullw%()"-Array zurückgepoked.

Jetzt könnte das so definierte Fenster mit

```
Closew Wi_handle%
Openw Wi_handle%
```

neu gezeichnet werden. Es geht aber komfortabler. AES enthält eine Funktion <Wind_set> (Sie haben diese beim DESKTOP schon lieben gelernt). Sie kann neue Fensterdaten übernehmen und zeichnet automatisch das Fenster entsprechend neu.

Unsere <Wind_set>-Routine will folgende Parameter übergeben bekommen: das Wi_handle%, einen Flag und die Koordinaten bzw. Elemente, deren Veränderung bewirkt werden soll.

Der Flag selektiert die gewünschten Veränderungen:

1 Ändert die Randelemente des Fensters. Diese werden im dritten Parameter übergeben, der vierte bis sechste erhalten eine 0.

2 Ändert den Namen des Fensters (werden wir nicht brauchen, weil Titlew dasselbe einfacher macht).

3 Ändert die Infozeile. Brauchen wir auch nicht.

5 Ändert die Größe des Fensters. Das ist das Flag, das in der Fullwindow-Routine benötigt wird.

8 Ändert die Position des horizontalen Sliders.

9 Ändert die Position des vertikalen Sliders.

10 Nach Eingabe eines neuen Wi_handle erzeugt AES die Message WM_TOPPED. Ich benutze diesen Flag-Wert nicht.

14 Legt eine neue Adresse für das Desktop an. Sie haben dieses Verfahren bereits kennengelernt.

15 Ändert die Größe des horizontalen Sliders.

16 Ändert die Größe des vertikalen Sliders.

Die Erläuterungen zu den Sliderflags siehe im Kapitel Slider.

Die nächste Meldung betrifft das Schließfeld. Bisher hat der Atari ja alles selber übernommen, Ihr Programm wurde kaum von den Fensterroutinen berührt. Nun aber sollten Sie sicherstellen, ob das Fenster wirklich geschlossen werden soll, ob eventuell Fensterinhalte gespeichert werden müssen oder ähnli-

ches. Deshalb wird zuerst in eine Routine gesprungen, die Sie selbst programmieren müssen. Alert-Boxen, Speicheralgorithmen usw. gehören dorthinein. Zurückgegeben werden muß aus dieser Prozedur ein Parameter mit dem Wert 1 für Beenden oder 0 für Weitermachen.

Sie sehen, unsere Fenster-Story besteht aus drei Elementen: Zum einen aus den hier beschriebenen Routinen, die - einmal programmiert - nicht wieder angerührt werden müssen, zum anderen aus den reinen Library-Routinen, also die Umsetzung der AES-Funktionen in GFA-BASIC, und zum dritten aus Prozeduren, die Sie für Ihr Programm jeweils neu spezifizieren müssen. Ich habe versucht, diesen dritten Bereich so einfach wie möglich zu halten.

Das Fenster wird geschlossen, aus dem Slot der vorherige Wi_handle-Wert ausgelesen und dann das Fenster mit dieser Handle-Nummer geöffnet.

Der Slider- und Pfeilbenutzung widme ich ein eigenes Kapitel, deshalb später mehr. Es bleibt nur noch das Vergrößern/ Verkleinern und das Bewegen des Fensters zu ermöglichen. Sie wissen, daß Menü(5) bis Menü(8) die neuen Koordinaten beinhalten. Übergeben wir diese doch einfach der Windtab-Tabelle und setzen mit <Wind_set>, Flag%=5, das veränderte Fenster. Genauso läuft's auch, womit wir die Beweglichkeit von Fenstern voll im Griff haben.

Eigentlich müßte das Schieben und das Verändern der Größen ja in zwei If..Endif-Bedingungen abgearbeitet werden. Einmal bleiben die X/Y-Koordinaten gleich, und es ändern sich nur Breite und Höhe des Fensters, das andere Mal ist es umgekehrt, die Breite und Höhe verändern sich nicht, die X/Y-Koordinaten jedoch wohl. Es ist eine Frage der Geschwindigkeit, ich meine jedoch, der Unterschied zwischen Aufteilung in zwei Bedingungen und der hier vorgestellten Zusammenfassung ist so minimal, daß er keine Rolle spielt.

```
********Library-Routinen**************
Procedure Window_init
  Dim Fullw%(4,4)
  Slot$=""
  On Menu Message Gosub Window_handling
  Let Fullw!=False
Return
'
Procedure Window_handling
  Local N%,A.%
  @Wind_update(1)
  If Menu(1)=20                              ! REDRAW
    @Redraw(Menu(4),Menu(5),Menu(6),Menu(7),Menu(8))
  Endif
  If Menu(1)=21                              ! TOPPED
    @Wind_find(*Wi_handle%)
    Openw Wi_handle%
    A.%=Instr(Slot$,Str$(Wi_handle%))
    Slot$=Left$(Slot$,A.%-1)+Mid$(Slot$,A.%+1)+Str$(Wi_handle%)
  Endif
  If Menu(1)=23                              ! FULLED
    If Fullw!=False
      For N%=1 To 4
        Let Fullw%(Wi_handle%,N%)=Dpeek(Windtab+Wi_handle%
                                                  *12-(10-2*N%))
        Dpoke Windtab+Wi_handle%*12-(10-2*N%),
                                        Dpeek(Windtab+50+(N%*2))
      Next N%
      Let Fullw!=True
    Else
      For N%=1 To 4
        Dpoke Windtab+Wi_handle%*12-(10-2*N%),Fullw%(Wi_handle%,N%)
      Next N%
      Let Fullw!=False
    Endif
    @Wind_set(Wi_handle%,5,Dpeek(Windtab+Wi_handle%*12-8),
              Dpeek(Windtab+Wi_handle%*12-6),
               Dpeek(Windtab+Wi_handle%*12-4),
                Dpeek(Windtab+Wi_handle%*12-2))
  Endif
  If Menu(1)=22                              ! CLOSED
    @Close_window(*Legal%)
    If Legal%=1
      Closew Wi_handle%
      Slot$=Left$(Slot$,Len(Slot$)-1)
      Wi_handle%=Val(Right$(Slot$,1))
```

```
    Endif
    Openw Wi_handle%
  Endif
  If Menu(1)=24                          ! ARROWED
  Endif
  If Menu(1)=25                          ! HSLID
  Endif
  If Menu(1)=26                          ! VSLID
  Endif
  If Menu(1)=27 Or Menu(1)=28            ! SIZED und MOVED
    For N%=1 To 4
      Dpoke Windtab+(Wi_handle%*12-(10-2*N%)),Menu(N%+4)
    Next N%
    @Wind_set(Wi_handle%,5,Menu(5),Menu(6),Menu(7),Menu(8))
  Endif
  @Wind_update(0)
Return
```

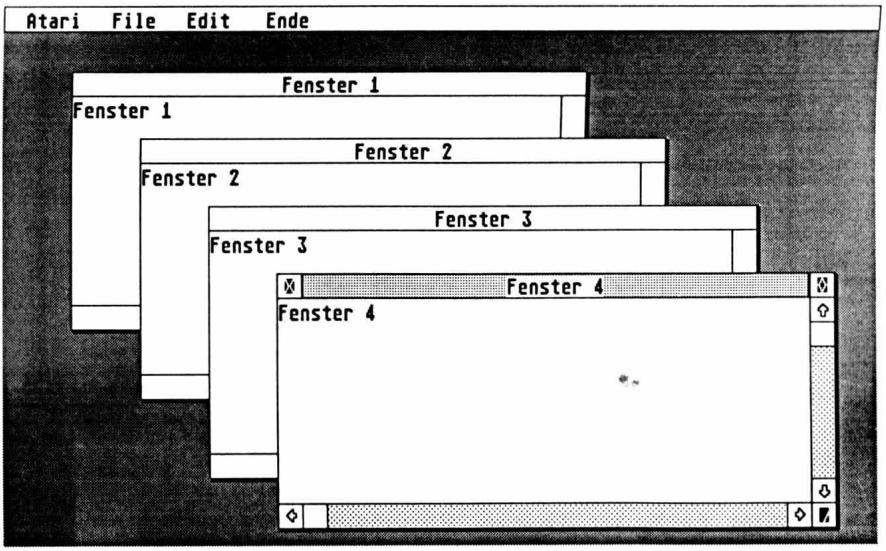


Abb. 25: **Fensterln**

Jetzt können Sie all das einmal ausprobieren. Sie werden sehen, es funktioniert prächtig. Nur - alles, was an Fensterteilen bei Verschiebungen in die Arbeitsbereiche der Fenster gelegt wird,

bleibt dort liegen, es wird nicht neu gezeichnet. Es wird also eine Routine benötigt, die auch das Regenerieren der Fensterarbeitszonen bewerkstelligt: REDRAW.

```
Gosub Window_init
Deffill 1,2,4
Pbox 0,0,640,400
Gosub Open_window("Fenster 1","",4095,50,50,200,200)
Gosub Open_window("Fenster 2","",4095,100,100,200,200)
Gosub Open_window("Fenster 3","",4095,150,150,200,200)
Gosub Open_window("Fenster 4","",4095,200,200,200,200)
Gosub Open_window("Fenster 5","",4095,200,200,100,100)
Do
  On Menu
  Exit If Mousek=2
Loop
For N%=0 To 4
  Closew N%
Next N%
'
Procedure Close_window(R%)
  Alert 0,"Wirklich schließen?",2,"Ja|nein",K%
  If K%=1
    *R%=1
  Else
    *R%=0
  Endif
Return
```

4.12.3 REDRAW-Routinen

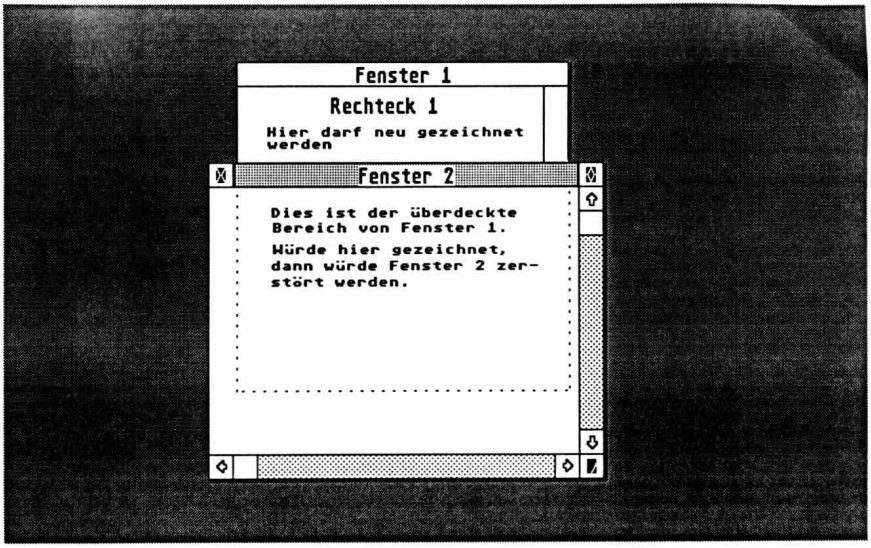
GEM besitzt einen klugen Verwaltungsapparat für das Regenerieren veränderter Bildschirmbereiche. Dieser Verwaltungsapparat arbeitet nach der Devise: Rationalisieren, wo's nur immer geht. Dazu wurde eine Unterabteilung eingerichtet, die sich um die Rechtecklisten kümmert. Sie wissen, daß Rationalisierung nicht unbedingt was mit Geschwindigkeit, sondern eher mit Resources-Optimierung zu tun hat. So ist es auch hier. Ein Verfahren, das den gesamten Bildschirm rettet und bei Bedarf neu zeichnet, wäre zwar schneller, würde aber immer mindestens 32 KByte verbrauchen. Die Rechteckliste ist langsamer, benötigt aber höchstens 10% des Speicherplatzes.

Sehen Sie sich die Abbildung an. Von den unterhalb des aktiven Fensters liegenden Fenstern brauchen immer nur Teilbereiche neu gezeichnet zu werden. Diese Teilbereiche können in einzelne Rechtecke untergliedert werden. Ein AES-interner Puffer enthält nun eine Liste all dieser Rechtecke, gegliedert nach den einzelnen Wi_handle. Wenn jetzt, Fenster für Fenster, diese Liste abgefragt, ein Clipping auf die Rechteckgrenzen gesetzt, der Fensterinhalt gezeichnet und das solange wiederholt wird, bis die Rechteckliste leer ist, dann ist das Ziel enorm rationell erreicht.

AES besitzt die Funktion <Wind_get>, die vom Grundsatz mit <Wind_set> korrespondiert, d.h., all das, was hier verändert werden soll, kann dort erst einmal nach seiner Wertigkeit abgefragt werden. Zusätzlich aber auch die Rechteckliste. Unsere GFA-Prozedur <Wind_get> hat ebenfalls 6 Parameter, das Wi_handle, einen Flag und vier Rückgabewerte.

Folgende Flags sind möglich:

4 Koordinaten des Arbeitsbereichs sind in den vier Rückgabeparametern enthalten.
5 Koordinaten der Gesamtgröße des Fensters (incl. des Randes) sind in den Rückgabeparametern enthalten.
6 Die Koordianten der Gesamtgröße des vorherigen Fensters sind in den Rückgabeparametern enthalten.
7 Die Koordinaten der maximalen Fenstergröße werden zurückgegeben.
8 Position des horizontalen Sliders
9 Position des vertikalen Sliders
10 Wi_handle% des aktiven Fensters im ersten Rückgabeparameter.
11 X/Y Koordinaten, Höhe und Breite des ersten Rechtecks innerhalb der Rechteckliste werden zurückgegeben
12 X/Y Koordinaten, Höhe und Breite des jeweils nächsten Rechtecks der Rechteckliste werden zurückgegeben.
15 Größe des horizontalen Sliders
16 Größe des vertikalen Sliders



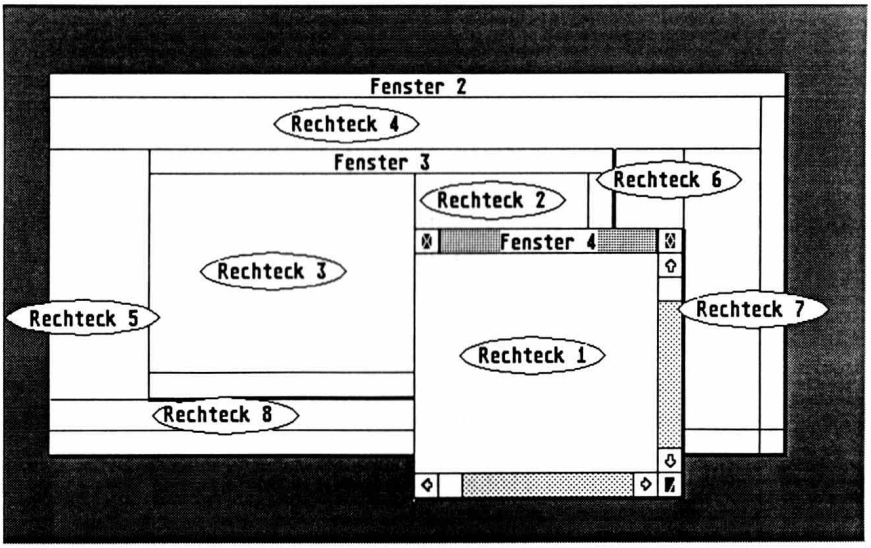


Abb. 26: **Rechtecke mit der REDRAW-Routine**

Für unsere Zwecke sind also die Flags 11 und 12 relevant. Fragen wir zunächst das erste Rechteck der Liste ab. Wir erhalten die X/Y-Koordinate der linken oderen Ecke und die Höhe und Breite. In einer While..Wend-Schleife mit der Bedingung "entweder die Breite oder die Höhe des Rechtecks müssen größer null sein", ein Rechteck muß also existieren, wird die Redraw-Routine abgearbeitet:

Aus den Werten der REDRAW-Message, die Auskunft darüber geben, welcher Bereich neu zu zeichnen ist, und denen des <Wind_get>-Aufrufs, der Auskunft gibt, welche Teile dieses Bereichs nicht durch andere Fenster überdeckt sind, wird das Clipping-Rectangle, in das gefahrlos gezeichnet werden darf, gebildet. Dies geschieht durch Übergabe dieser Werte an die VDI-Routine Clipping_rectangle, die automatisch alle Zeichnungen auf einen Bereich begrenzt, sowie Verlegen des Nullpunkts für Grafikbefehle in die linke obere Ecke des geclippten Rechtecks. Dies kann einfach durch Poken dieser Werte in das 64. und 66. Element der Windtab-Tabelle geschehen.

Nachdem das Programm jetzt also weiß, wo es zu regenerieren hat, muß es noch wissen, was es denn regenerieren soll. Und das müssen Sie wieder selber machen. In einer User-Routine schreiben oder malen Sie einfach den gesamten Bildschirminhalt neu, das Clippen besorgt den Ausschnitt.

Anschließend wird <Wind_get> mit dem Flag 12 aufgerufen, das die Koordinaten des nächsten Rechtecks zurückgibt.

Das sieht alles einigermaßen langsam aus. Wenn man dann erfährt, daß diese Redraw-Routine für jedes neu zu zeichnende Fenster erneut angelaufen wird, jeweils mit dem entsprechenden Wi_handle, kann einem schon schwindlig werden: Denn die Dauer des Fensteraufbaus ist zwar spürbar, aber trotz der irrwitzigen Rechnereien leistet der 68000 und sein Betriebssystem hier sein Meisterstück.

```
Procedure Redraw(Wi.h%,R.x%,R.y%,R.w%,R.h%)
  Local Gr_x%,Gr_y%,Ret%,Rx_%,Ry_%,Rw_%,Rh_%,Tb%,Th%,Tx%,Ty%
  @Wind_get(Wi.h%,11,*Rx_%,*Ry_%,*Rw_%,*Rh_%)
  While Rw_%+Rh_%<>0
```

```
    If Rw_%>0 And Rh_%>0
      @Clip_rect(Rx_%,Ry_%,Rw_%,Rh_%)
      @Wind_get(Wi.h%,4,*Gr_x%,*Gr_y%,*Gr_w%,*Gr_h%)
      Dpoke Windtab+64,Gr_x%
      Dpoke Windtab+66,Gr_y%
      Gosub Redraw_manuell(Wi.h%,Gr_w%,Gr_h%)
    Endif
    @Wind_get(Wi.h%,12,*Rx_%,*Ry_%,*Rw_%,*Rh_%)
  Wend
  @Wind_set(0,10,Wi_handle%,0,0,0)
Return
```

Dies ist die komplette Redraw-Routine. Für Sie als Programmierer gibt's noch die "User-Prozedur", in die ich hier nur exemplarisch Fensterinhalte eingetragen habe. Da wir hier mit Clipping arbeiten, funktionieren Cls oder Clearw zum Löschen des Bildschirms nicht. Deshalb muß eine reine VDI-Funktion (Pbox) herangezogen werden. Diese hat jedoch den Nachteil, daß sie immer einen Rahmen zeichnet. Dieser kann nur mit einer weiteren VDI-Funktion ausgeschaltet werden: <Set_perimeter>. Flag!=0 schaltet den Rahmen aus, Flag!=-1 schaltet ihn wieder an.

```
********Library_routine************
Procedure Rahmen(F!)
  Dpoke Intin,F!
  Dpoke Contrl+2,0
  Dpoke Contrl+6,1
  Vdisys 104
Return
'
Procedure Redraw_manuell(Wi_h%,dummy%,dummy%)
  Deffill 1,0
  Gosub Rahmen(0)
  Pbox 0,0,640,400
  If Wi_h%=1
    Print At(1,1);"Fenster 1"
  Endif
  If Wi_h%=2
    Print At(1,1);"Fenster 2"
  Endif
  If Wi_h%=3
    Print At(1,1);"Fenster 3"
  Endif
```

```
  If Wi_h%=4
    Print At(1,1);"Fenster 4"
  Endif
Return
```

4.12.4 Slider und Arrows

Zum Schluß unserer Betrachtungen über Fenster soll anhand der Schieber und Pfeile gezeigt werden, wie diese bedient werden. Viel interessanter sind jedoch kurze Scroll-Routinen, die für Textverarbeitungen, Editoren usw. aufgebaut werden können. Eine solche Routine folgt in Kürze. Sie funktioniert exakt so, wie hier beschrieben, auch in den komplexesten Editoren. Zuerst aber wie üblich etws zum Verständnis.

Der Event_handler gibt drei verschiedene Scroll_messages aus: VSLID und HSLID beim Verschieben der vertikalen oder horizontalen Slider und WM_ARROWED beim Anklicken einer der vier Pfeile bzw. in eines der grauen Felder ober- bzw. unterhalb der Slider.

Insgesamt sind damit also 10 verschiedene Meldungen selektierbar, die folgende Ereignisse auslösen sollten:

Verschieben des Bildschirminhalts nach oben bzw. unten

- um eine Zeile
- um eine Seite
- um einen bestimmbaren Offset.

Dasselbe gilt natürlich für die Verschiebungen nach rechts und links.

Das Menu-Array gibt im Menü(1) die Indizes 24 für WM_ARROWED, 25 für WM_HSLID und 26 für WM_VSLID. Menü(5) spezifiziert dann: bei V_SLID und H_SLID wird die Slider-Position ausgegeben, bei WM_ARROWED der Index des angewählten Fensterelements:

0	*PG_UP*	Seite nach oben
1	*PG_DOWN*	Seite nach unten
2	*ARROW_UP*	Zeile nach oben
3	*ARROW_DOWN*	Zeile nach unetn
4	*PG_LEFT*	Seite nach links
5	*PG_RIGHT*	Seite nach rechts
6	*ARROW_LEFT*	Zeichen nach links
7	*ARROW_RIGHT*	Zeichen nach rechts

GEM hat nicht vorgegeben, was eine Zeile oder eine Seite ist. Sie können eine Scan-Linie, also eine Pixel-Reihe als Zeile vorsehen, eine Schriftzeile im 16-Pixel-Raster, eine im 8-Pixel-Raster, halbe Zeilen, Sinnvolles oder auch Quatsch. Ebenso kann eine Seitenlänge vom Programmierer definiert werden.

Sie erhalten sowohl bei Größen- als auch bei Positionsveränderungen die Mitteilung, daß die Slider-Balken (egal, wie groß das Fenster ist) eine Größe und eine Position zwischen 0 (oben bzw. links) und 1000 (unten bzw. rechts) haben.

Damit gelten folgende Formeln:

Slider-Größe	1000/(Gesamtgröße/sichtbarer Teil)
Slider-Position	1000/(Gesamtgröße/erste Fensterzeile)

Ein Text habe 80 Zeilen, davon sind die Zeilen 30 bis 50 auf dem Bildschirm sichtbar:

Slider-Größe	1000/80/20 250
Slider-Position	1000/80/30 375

Nach Auflösen der Formel bekommt man die erste Fensterzeile:

erste Fensterzeile = (Slider-Position*Gesamtgröße)/1000

Die Ergebnisse der ersten beiden Formeln sind unserem bekannten Wind_set zu übergeben, das die Randzonen des Fensters dann entsprechend korrigiert. Das Ergebnis der dritten Formel muß Ihrer User-Routine übergeben werden, damit der Text entsprechend gescrollt wird.

Nun aber in die Praxis:

Das Window_handling in der vorherigen Fensterroutine war ja in den If..Endif-Verzweigungen zu Slider-Meldungen noch nackt und bloß. Schreiben Sie wie folgt:

```
' Programm: FENSTER2.BAS
'
Procedure Window_handling
  @Wind_update(1)
  If Menu(1)=24 ! ARROWED
    @Wind_get(Wi_handle%,4,*Wi_x%,*Wi_y%,*Wi_w%,*Wi_h%)
    @Slid(Menu(5),*A.%,*T.%,*St.%)
    @Slider(Wi_handle%,A.%,T.%,St.%)
    @Redraw(Wi_handle%,Wi_x%,Wi_y%,Wi_w%,Wi_h%)
  Endif
  If Menu(1)=25 ! HSLID
    @Wind_get(Wi_handle%,4,*Wi_x%,*Wi_y%,*Wi_w%,*Wi_h%)
    @Slid(Menu(5)+1000,*A.%,*T.%,*St.%)
    @Slider(Wi_handle%,A.%,T.%,St.%)
    @Redraw(Wi_handle%,Wi_x%,Wi_y%,Wi_w%,Wi_h%)
  Endif
  If Menu(1)=26 ! VSLID
    @Wind_get(Wi_handle%,4,*Wi_x%,*Wi_y%,*Wi_w%,*Wi_h%)
    @Slid(Menu(5)+2000,*A.%,*T.%,*St.%)
    @Slider(Wi_handle%,A.%,T.%,St.%)
    @Redraw(Wi_handle%,Wi_x%,Wi_y%,Wi_w%,Wi_h%)
  Endif
  @Wind_update(0)
Return
```

Die drei Routinen, die das Sliderhandling verwalten, sind fast identisch: Es wird erst einmal wieder Wind_get nach der momentanen Größe des Fensterarbeitsbereichs abgefragt. Der Index aus Menu(5), also entweder das eben bediente Randelement oder die neue Position der Slider wird der User-Prozedur Slid übergeben. Ich habe hier etwas herumgetrickst. Damit für Sie alles an Slider-Informationen innerhalb einer Prozedur abgearbeitet werden kann, kommen die Indizes zwischen 0 und 7 aus der WM_ARROWED-Message direkt herüber, die H_SLID- und V_SLID-Meldungen habe ich für den V_SLID mit einem Off-

set von 1000 und für den H_SLID mit einem Offset von 2000 versehen. Damit sind garantiert exakt ausfilterbare Werte in der User-Routine zu finden.

Die <Procedure Slider> rechnet die aus der User-Routine zurückkommenden Werte für "Alles" (A.%), "Teil" (T.%) und "ersteZeile" (St.%) entsprechend der obigen Formeln in Slider-Koordinaten um und übergibt diese an Wind_set: einmal mit dem Index 9 für die Größe und einmal mit dem Index 16 für die Position. In der <Procedure Redraw> wird der Inhalt des Fensterarbeitsbereichs dann wie bekannt neu gezeichnet.

```
Procedure Slider(W.h%,Alles%,Teil%,Screenanfang%)
  Sl_pos%=1000/(Alles%/Screenanfang%)
  Sl_groesse%=1000/(Alles%/Teil%)
  @Wind_set(W.h%,16,Sl_groesse%,0,0,0)
  @Wind_set(W.h%,9,Sl_pos%,0,0,0)
Return
```

Das Beispiel für diese Library-Routinen, das ich Ihnen jetzt vorstelle, könnte einer Textverarbeitung als Scrollroutine dienen. Das Fenster wird geöffnet, ein Text von 100 Zeilen wird simuliert und zwei Variablen <Top_line%> für die erste Bildschirmzeile und "Base_line%" für die letzte Textzeile mit den entsprechenden Werten belegt.

```
Gosub Open_window("Fenster 1","",4095,50,50,300,300)
'
Dim Text$(100)
Top_line%=1
Base_line%=100
For N%=Top_line% To Base_line%
  Let Text$(N%)="Dies ist Zeile "+Str$(N%)
Next N%
'
Do
  On Menu
  Exit If Mousek=2
Loop
Closew 1
```

Zwei User_routinen managen das Scrolling: "Redraw" habe ich Ihnen schon vorgestellt, hier ist sie entsprechend dieser Aufgabe

ausgearbeitet, d.h. sie zeigt immer die der Fenstergröße entsprechende Länge mit dem der Slider-Position entsprechenden Anfang des Textes.

Die <Procedure Slid> verändert den Wert der Variablen Top_line%: Seitenverschiebungen verändern die Top_line% um die Anzahl der sichtbaren Zeilen, Zeilenverschiebungen reduzieren bzw. erhöhen sie um 1. Bei V_SLID und H_SLID wird entsprechend der obigen Formel die neue Top_line% errechnet - natürlich nicht, bevor der Trickoffset von 1000 bzw. 2000 wieder subtrahiert wurde.

Zurückgegeben wird immer die jeweilige Gesamtgröße des Dokuments, meistens eine globale Variable, der sichtbare Teil des Dokuments, abhängig von der momentanen Fenstergröße und die eben errechnete Top_line%, also die erste sichtbare Dokumentenzeile. In dieser Weise ist die Userprozedur für fast alle Anwendungen gleichermaßen zu benutzen.

Ich habe Ihnen nur die Routinen für das vertikale Scrolling vorgeführt. Klar, daß absolut kein Unterschied bei den horizontalen Scroll-Balken zu melden ist, oder?

```
Procedure Slid(F%,Ret1%,Ret2%,Ret3%)
  Local H.oehe%
  H.oehe%=Int(Wi_h%/16)
  If F%=0                          ! Pg_up
    Sub Top_line%,H.oehe%
  Else
    If F%=1                        ! Pg_dn
      Add Top_line%,H.oehe%
    Else
      If F%=2                      ! Up_arrow
        Dec Top_line%
      Else
        If F%=3                    ! Dn_arrow
          Inc Top_line%
        Else
          If F%=4                  ! Pg_lf
          Else
            If F%=5                ! Pg_rt
            Else
              If F%=6              ! Lf_arrow
```

```
                Else
                  If F%=7                  ! Rt_arrow
                  Else
                    If F%>=2000            ! VSLID
                      Sub F%,2000
                      Top_line%=Base_line%*(F%/1000)
                    Else
                      If F%>=1000          ! HSLID
                        Sub F%,1000
                      Endif
                    Endif
                  Endif
                Endif
              Endif
            Endif
          Endif
        Endif
      Endif
    Endif
  Endif
  If Top_line%<1
    Top_line%=1
  Else
    If Top_line%>Base_line%
      Top_line%=Base_line%
    Endif
  Endif
  *Ret1%=Base_line%
  *Ret2%=H.oehe%
  *Ret3%=Top_line%
Return
'

Procedure Redraw_manuell(Wi_h%,Zeichen%,Zeilen%)
  If Wi_h%=1
    Deffill 1,0
    Gosub Rahmen(0)
    Pbox 0,0,640,400
    Zeilen%=Int(Zeilen%/16)
    If Top_line%+Zeilen%>Base_line%
      Top_line%=Base_line%-Zeilen%
    Endif
    For N%=Top_line% To Top_line%+Zeilen%
      Print At(1,N%-Top_line%+1);Text$(N%);
    Next N%
  Endif
Return
```

4.12.5 Sieben Fenster? Das geht doch gar nicht!

Gemach! Es geht wohl. Auch wenn Sie's sicher selten, vielleicht nie gebrauchen werden, möchte ich Ihnen die Möglichkeit nicht vorenthalten, insgesamt bis zu sieben Fenster gleichzeitig offenzuhalten. Aber: Wirklich nur sieben. Ein Deskaccessory, das ein neues Fenster öffnen möchte, wird unweigerlich Fehlermeldungen, vielleicht sogar Bomben produzieren. Und noch eine Warnung ist angebracht. Die Digital-Research-Leute haben sich mit dieser GEM-Version viel Mühe gemacht. Aber einiges ist dennoch einigermaßen schnodderig programmiert. So auch der Umgang mit den Fenster-Handles. Nicht immer geht alles so glatt, wie man es gerne möchte. Ein Fehlerchen bei der Programmentwicklung gemacht, und sang- und klanglos geht der Atari in die Knie. Wenn Sie Glück haben, können Sie gerade noch abspeichern, meistens wohl auch das nicht mehr. Also, seien Sie aufmerksam, die Reset-Taste und Ihre Nerven werden es Ihnen danken.

Mit seiner speziellen Fensterroutine über die Windtab-Tabelle hat Frank Ostrowski die Fensteranzahl auf 4 begrenzt. Um mehr zu bekommen, müssen wir direkt in die AES-Funktionen hineingehen und Fenster so öffnen, wie auch die C- und Pascal-Programmierer es machen müssen.

Das geht in drei Schritten vor sich:

Mit Hilfe der Funktion <Wind_create> wird Speicherplatz für das Fenster reserviert, das Handle zurückgegeben und die maximal mögliche Fenstergröße in den Koordinatenparametern sowie die Bedienelemente im W.f%-Parameter bestimmt.

```
Procedure Wind_create(W.ih%,W.f%,W.x%,W.y%,W.w%,W.h%)
  Dpoke Gintin,W.f%
  Dpoke Gintin+2,W.x%
  Dpoke Gintin+4,W.y%
  Dpoke Gintin+6,W.w%
  Dpoke Gintin+8,W.h%
  Gemsys 100
  *W.ih%=Dpeek(Gintout)
Return
```

Danach wird mit <Wind_set> mit Hilfe der Flags 2 und 3 der Fenstertitel sowie der Inhalt der Info-Zeile übergeben. Das läuft nun aber etwas anders ab als in der Ihnen bekannten <Wind_set>-Version, weil Name und Infozeile als Adresse und damit als Longword gebraucht werden. Vergessen werden darf nicht die Konvention des 0-Byte am Stringende.

```
Procedure Wind_set(W.ih%,W.f%,W.x%,W.w%,W.h%)
  Dpoke Gintin,W.ih%
  Dpoke Gintin+2,W.f%
  Lpoke Gintin+4,W.x%
  Gemsys 105
Return
```

Das Sesam-öffne-dich ist jetzt die neue AES-Funktion <Wind_open>, die das wi-handle und die wirklichen Fensterausmaße übergeben bekommen muß.

```
Procedure Wind_open(W.ih%,W.x%,W.y%,W.w%,W.h%)
  Dpoke Gintin,W.ih%
  Dpoke Gintin+2,W.x%
  Dpoke Gintin+4,W.y%
  Dpoke Gintin+6,W.w%
  Dpoke Gintin+8,W.h%
  Gemsys 101
Return
```

Jetzt funktioniert aber nichts mehr von den bekannten Ostrowskischen Fensterroutinen. Angefangen beim Hintergrundaufbau des Fensterarbeitsbereichs bis hin zum endgültigen Schließen der Fenster darf jetzt nur nun mit reinen AES-Funktionen gearbeitet werden.

Das Löschen geschieht mit Hilfe zweier Gemsys-Aufrufe. Der eine, <Wind_close>, löscht das Fenster vom Bildschirm, behält aber das Handle und den Speicherplatz, der andere, <Wind_delete>, löscht das Handle und den Speicherplatz, jedoch nicht die Bildschirmzeichnung. Ich habe beide Aufrufe in einer Prozedur zusammengeschlossen, weil es wohl kaum Gründe gibt, das eine nicht ohne das andere zu tun.

```
Procedure Wind_delete(W.ih%)
  Dpoke Gintin,W.ih%
  Gemsys 102
  Gemsys 103
Return
```

Der Bildhintergrundaufbau, also das "Clearw", wird mit Hilfe des Pbox-Befehls gemacht, der aus <Wind_get> die genauen Größenparameter erhält. Die Rahmenprozedur ist bekannt.

```
Procedure Wind_clear(Wi.h%,W.x%,W.y%,W.w%,W.h%)
  Deffill 1,0
  Gosub Rahmen(-1)
  Pbox W.x%-1,W.y%-1,W.x%+W.w%,W.y%+W.h%
  Gosub Rahmen(0)
Return
```

Und damit genug der Vorrede. Mit dem Listing "Fenster3.Bas" werden jetzt 7 Fenster gezeichnet:

```
Wx%=50
Wy%=50
Do
  @Wind_create(*Wi_handle%,&X111111111111,0,19,640,381)
  Exit If Wi_handle%<0 Or Wi_handle%>10
  N.ame$="Testfenster"+Chr$(0)
  @Wind_set(Wi_handle%,2,Varptr(N.ame$),0,0)
  I.nfo$="Dies ist Fenster Nr. "+Str$(Wi_handle%)+Chr$(0)
  @Wind_set(Wi_handle%,3,Varptr(I.nfo$),0,0)
  @Wind_open(Wi_handle%,Wx%,Wy%,400,200)
  @Wind_get(Wi_handle%,4,*X%,*Y%,*W%,*H%)
  @Wind_clear(Wi_handle%,X%,Y%,W%,H%)
  Add Wx%,10
  Add Wy%,10
Loop
Do
  Exit If Mousek
Loop
For N%=1 To 7
  @Wind_delete(N%)
Next N%
End
```

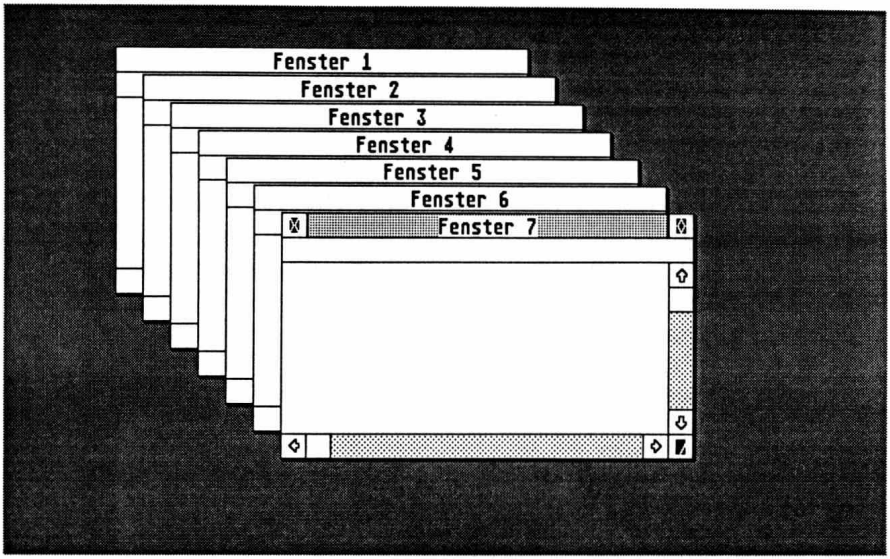


Abb. 27: 7 Fenster

4.13 Tutti-Frutti: Algorithmen für einen Resources-Baukasten

Hier könnte ja ein schönes, komplettes Prograämmchen stehen, das die Objektkonstruktionen für GFA-BASIC aufbereitet, ein Resource-Construction-Kit wie das RCS, nur eben auf unser Programmiermedium GFA-BASIC zugeschnitten. Aus rechtlichen Gründen solls nicht sein, also gibt's nur den Hauptgang, Vorsuppe und Nachspeisen müssen Sie selber drumherumkochen. Vielleicht gibt's das RCK ja bald in der Data-Welt-Edition?

Aus diesem Grund bitten wir Sie auch, unmögliche Eingaben zu unterlassen, da diese von der Grundversion des Programms nicht abgefangen werden. Hüten Sie sich also davor, die Objekte über die Grenzen der Wurzelbox hinaus zu bewegen oder zu vergrößern. Bei der von Ihnen selbst vervollständigten Version sollten Sie diese Möglichkeiten abfangen.

Sie wissen jetzt schon soviel über die Datenstrukturen von GEM-Objekten, daß vorerst eigentlich nicht mehr viel gesagt werden muß. Deshalb die Beschränkung auf das Wesentliche:

Unser Resources-Baukasten erscheint in einem bildschirmgroßen Fenster. Zunächst ist nichts sichtbar außer dem Wurzelobjekt, eine BOX mit etwas dickerem Rand. Mit der HELP-Taste bekommen Sie ein Formular (hier allerdings nicht GEM, sondern handgemacht) auf den Bildschirm, mit dessen Hilfe Sie das einzugebende neue Objekt spezifizieren können. Nach einem Klick auf die linke Maustaste erscheint das Objekt, Sie können es dann mit der Maus dorthin plazieren, wo Sie es haben wollen. Verschiebungen und Vergrößerungen mittels der Graf_library sind natürlich jederzeit möglich.

Im Grundbaukasten sind weder IMAGES noch ICONs enthalten. BOXEN, STRINGs und TEXTE sind selbstverständlich definierbar. Ich denke aber, mit Hilfe dieses Grundbaukasten werden Sie das darüber hinaus Wünschenswerte problemlos selbst basteln können.

```
' ****************************************
' *  Resources-Baukasten: Grundelemente *
' ****************************************
'
Rsc$=Space$(10000)                    ! Der Speicherplatz für den
Rsc_adr%=Varptr(Rsc$)                 ! RSC-File
'
Object$=Space$(100*24)                ! Speicherplatz für die
Tree%=Varptr(Object$)                 ! Objectstruktur
'
Tedinfo$=Space$(20*28)                ! Speicherplatz für die
Tedinfo_adr%=Varptr(Tedinfo$)         ! Tedinfostruktur
'
Rs_string$=Space$(1000)               ! Speicherplatz für die
String_adr%=Varptr(Rs_string$)        ! Strings und Texte
'
Dim Ind$(100)                         ! 100 Objekte
Tree_counter%=0                       ! laufende Zähler
String_counter%=0
Ted_counter%=0
Index%=0
```

```
Ted_index%=0
'
Titlew 1,""                                  ! ein bildschirmgroßes
Gosub Open_window(1,0,0,19,639,380)          ! Fenster wird eröffnet
'
' Objektstruktur der Wurzel
Gosub Def_obspec(0,2,1,1,1,0)
Gosub Make_object(20,0,&H10,Ob_spec%,0,22,500,300)
On Menu Key Gosub Kreuzung
```

Der vorgestellte Baukasten geht davon aus, daß die Benutzerführung von Ihnen komplettiert wird. Deshalb werden hier die einzelnen Objektdefinitionen per Tastatur in eine Box eingegeben, die alle Eintragsvarianten ermöglicht. Gestartet wird die entsprechende Prozedur, also das Einfügen neuer Objekte in den Baum, durch Betätigen der Help-Taste.

4.13.1 Objekte bewegen

Die Hauptverzweigungen erfolgen innerhalb einer Do..Loop-Schleife, die abfragt, ob der Mauszeiger über einem Objekt steht und über welchem. Nach Anklicken der linken Maustaste kann dieses Objekt in der rechten unteren Ecke vergrößert oder verkleinert und am oberen Rand verschoben werden. Außerdem wird beim Anwählen der Help-Taste in der <Procedure Kreuzung> das Generieren eines neuen Objekts eingeleitet, mit der Funktionstaste 10 geht's ab zum Speichern des RSC-Files.

```
Do
  On Menu
  Gosub Objc_find(*Ob%)                      ! Damit wir immer wissen, wo
  '                                          ! wir sind!
  While Mousek=1                             ! Maustastendruck zum Ver-
    Mouse Mx%,My%,K%                         ! schieben oder Vergrößern der
    For N%=0 To Index%                       ! Objekte. Der Baum wird nach
      X.t%=Dpeek(Tree%+(24*N%)+16)           ! Koordinaten durchsucht. Bei
      Y.t%=Dpeek(Tree%+(24*N%)+18)           ! Übereinstimmung von Objekt-
      W.t%=Dpeek(Tree%+(24*N%)+20)           ! Koordinaten und Mauszeiger
      H.t%=Dpeek(Tree%+(24*N%)+22)           ! passiert folgendes:
      '
```

```
      If Mx%>X.t%+W.t%-5 And Mx%<X.t%+W.t%+5 And My%>Y.t%+H.t%-5
                                                 And My%<Y.t%+H.t%+5
        Gosub Graf_rubberbox(X.t%,Y.t%+19,*Gw.t%,*Gh.t%)
        Dpoke Tree%+(24*N%)+20,Gw.t%      ! Wenn die rechte untere Ecke
        Dpoke Tree%+(24*N%)+22,Gh.t%      ! angeklickt wurde, dann ver-
        Gosub Objc_draw                   ! größern, neue Werte einpoken
      Endif                               ! und neu zeichnen.
      If Mx%>X.t% And Mx%<X.t%+W.t% And My%>Y.t%-3 And My%<Y.t%+3
        Gosub Graf_dragbox(X.t%,Y.t%+19,W.t%,H.t%,*Gx.t%,*Gy.t%)
        Dpoke Tree%+(24*N%)+16,Gx.t%      ! Wenn oberer Rand angeklickt
        Dpoke Tree%+(24*N%)+18,Gy.t%      ! wurde, dann verschieben, neue
        Gosub Objc_draw                   ! Werte einpoken und neu
      Endif                               ! zeichnen.
    Next N%
  Wend
Loop
'
Procedure Kreuzung
  If Menu(14) Div 256=68                ! Funktionstaste 10
    Gosub Save_rsc                      ! Abspeichern
  Endif
  If Menu(14) Div 256=67                ! Ist nicht eingebaut
    Gosub Load_rsc
  Endif
  If Menu(14) Div 256=98                ! HELP-Taste, es geht
    Gosub Define_object                 ! um ein neues Objekt
  Endif
Return
```

4.13.2 Objekte definieren

Diese nächste Prozedur ist einigermaßen langweilig, zugegeben. Innerhalb eines Rahmens ist eine Maske vorgegeben, die Sie mit Hilfe des <Form_input> zu füllen haben. Ich denke, an diesem Punkt ist Ihrer Kreativität keine Grenze gesetzt, es liegen ihr aber auch keine nicht überwindlichen Hindernisse im Weg.

Zu definieren sind folgende Strukturwerte von Objekten, Tedinfos und Strings:

- Der Typ des Objekts als Zahlenwert zwischen 20 und 31.
- Die Flags des Objekts als Zahlenwert zwischen 0 und 256.

- Der Anfangsstatus des Objekts, und zwar als Zahlenwert zwischen 0 und 32.
- Eventuell der Charakter, den ein Objekt "BOXCHAR" enthalten soll.
- Rand und Füllmuster für das Objekt, wenn es keinen Zeiger auf andere Strukturen enthält. Der Rand als Zahl zwischen 0 und 256, das Muster als Zahl zwischen 0 und 15.
- Falls das Objekt vom Typ STRING oder BUTTON ist, muß der String, der enthalten sein soll, übergeben werden können. Er muß ein 0-Byte als Abschluß haben.
- Und last but not least die aus Text, Maske und Gültigkeitsangaben zusammengesetzten Texte bei Objekten mit Zeiger auf eine TEDINFO-Struktur. Diese wollen ein bißchen sensibel behandelt werden: Der "Text" ist nur ein Eintrag, außer dem abschließenden 0-Byte kann er so bleiben wie eingegeben.

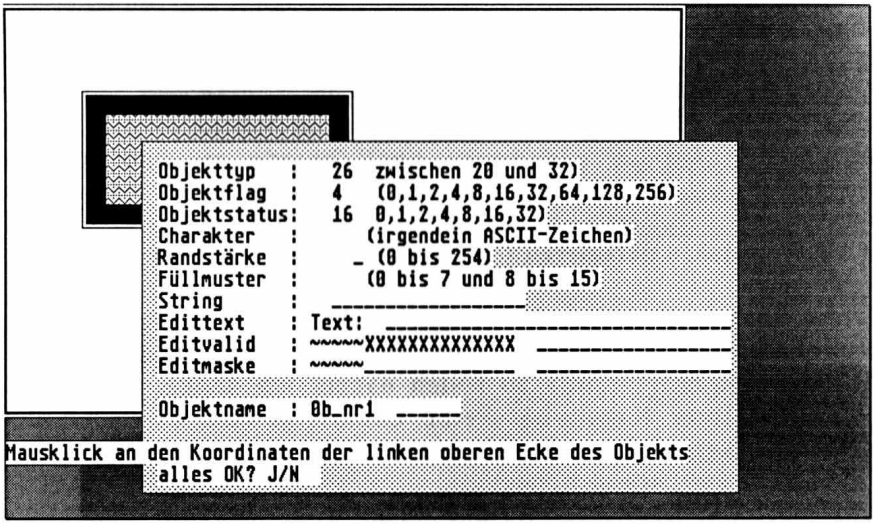


Abb. 28: Resources-Baukasten

Die Validation-Einträge bestehen ausschließlich aus vordefinierten Zeichenfolgen. Da alle RCS den Standard gesetzt haben, die Welle "~" als Platzhalter zu nehmen, wird dieser wieder herausgefiltert, so daß am Ende wirklich nur die Validation-Parameter übrigbleiben. Der übrigbleibende Rest des Validation-Strings bestimmt die Maskenlänge, die in den restlichen Programmzeilen dahingehend begrenzt wird.

Der Name des Objekts kann eingegeben werden, muß aber nicht, und nach der Sicherheitsabfrage geht's zum Aufbau der Objektstruktur.

```
' Programm: RCS_KIT.BAS
'
Procedure Objektdefinitionen
  Sget Screen$
  Deffill 1,2,1
  Pbox 100,100,540,360
  Print At(15,8);"Objekttyp    :   __ (zwischen 20 und 32)"
  Print At(15,9);"Objektflag   :   ___ (0,1,2,4,8,16,32,64,128,256)"
  Print At(15,10);"Objektstatus:   __ (0,1,2,4,8,16,32)"
  Print At(15,11);"Charakter   :   _ (irgendein ASCII-Zeichen)"
  Print At(15,12);"Randstärke  :   ___ (0 bis 254)"
  Print At(15,13);"Füllmuster  :   __ (0 bis 7 und 8 bis 15)"
  Print At(15,14);"String      : ____________________"
  Print At(15,15);"Edittext    : ______________________________"
  Print At(15,16);"Editvalid   : ______________________________"
  Print At(15,17);"Editmaske   : ______________________________"
  Print At(15,19);"Objektname  : _____________"
  '
  Start:
  Print At(31,8);
  Form Input 2,Typ$            ! Ob_type
  Typ%=Val(Typ$)
  Print At(31,9);
  Form Input 3,Flag$           ! Ob_flag
  Flag%=Val(Flag$)
  Print At(31,10);
  Form Input 2,Statu$          ! Ob_state
  Status%=Val(Statu$)
  Print At(31,11);
  Form Input 1,Char$           ! Ob_char
  Chr=Asc(Char$)
  Print At(31,12);
```

```
  Form Input 3,Rand$
  Rand%=Val(Rand$)              ! Rand für Ob_spec
  Print At(31,13);
  Form Input 2,Fuell$
  Muster%=Val(Fuell$)           ! Muster für Ob_spec
  Print At(29,14);
  Form Input 15,String$
  String$=String$+Chr$(0)       ! Rs_string
  '
  Print At(29,15);
  Form Input 39,E.text$
  E.text$=E.text$+Chr$(0)
  '
  Print At(29,16);
  Form Input 39,E.valid$
  Temp$=Space$(Len(E.text$)-1)
  Lset Temp$=E.valid$
  For N%=1 To Len(Temp$)
    Exit If Mid$(Temp$,N%,1)<>"~"
  Next N%
  E.valid$=Mid$(Temp$,N%)+Chr$(0)
  '
  Print At(29,17);
  Form Input 39,E.mask$
  Temp$=Space$(Len(E.text$)-1)
  Lset Temp$=E.mask$
  E.mask$=Mid$(Temp$,N%)+Chr$(0)
  '
  Print At(29,19);
  Form Input 14,Ind$(Index%)    ! Objektname
  Print At(15,22);
  Input "alles OK? J/N",A$
  If A$="n"
    Goto Start
  Endif
  Sput Screen$
Return
```

4.13.3 Objekte spezifizieren

Nun wird's schon spannender. Uns fehlen noch einige Objektspezifikationen. Zum einen natürlich die Koordinaten. Breite und Höhe werden zunächst vorgegeben, sie können ja mit

Hilfe der Graf_library jederzeit verändert werden. Die X%- und Y%-Koordinaten werden mittels eines Mausklicks geliefert.

Nun wird noch die Ob_spec-Definition benötigt. Sie wissen: Für unterschiedliche Objekttypen müssen unterschiedliche Ob_specs geliefert werden. Bei BOX, IBOX und BOXCHAR wird ein Longword mit den Erläuterungen zu Rand, Muster etc. gewünscht. Um diese zu liefern, ist nun unsere alte Def_obspec-Routine bestens geeignet.

BUTTON und STRING-Objekte wollen die Adresse des Strings haben. Wie Sie wissen, liegt diese in einem separaten Speicherraum, hintereinander aufgereiht, aber gemischt mit den Stringinformationen zu den TEDINFO-Strukturen. Was tun? Erst einmal sorgen wir dafür, daß die Länge des Strings geradzahlig ist, damit wir keine Peek-oder Poke-Fehlermeldung erhalten. Bei Bedarf wird also einfach ein Chr$(0) angehängt. Dann wird mittels des Bmove-Befehls der String ab Anfangsadresse bis zu seiner Gesamtlänge in den nächsten freien Platz des (bekanntlich reservierten) Stringspeicherraumes verschoben. Dieser freie Platz wird durch den Zähler <String_counter%> verwaltet. Die Objektspezifikation Ob_spec ist demnach die absolute Adresse des Strings.

TEXT, BOXTEXT, FTEXT und FBOXTEXT- Objekte benötigen in Ob_spec die zugehörige TEDINFO-Adresse, in der wiederum die Adresse der drei Strings erwartet wird. Diese drei Strings fassen wir zu einem zusammen (das können wir ja, weil sie auch zusammenhängend in den Stringspeicher eingebaut werden) und begradigen die Länge. Das Verschieben in den Stringspeicher läuft wie oben. In der <Procedure Tedinfo> wird nun diese Struktur gefüllt. Der nächste freie Speicher - absolut! - wird errechnet aus der TEDINFO-Anfangsadresse plus dem Verwalter Ted_counter%. Diese Struktur kennen Sie, ich gehe nur in zwei Punkten darauf ein: Font% und Just% habe ich der Einfachheit halber vorbelegt, Sie können diese Werte natürlich ebenfalls veränderlich machen. Auch die Musterfarbe ist vorbelegt, für sie gilt dasselbe. Die in die ersten drei Longwords einzupokenden Adressen sind: der Textanfang, also der Anfang des zugehörigen Strings, der Maskenanfang, also der Anfang des zu-

gehörigen Strings zuzüglich der Länge des Textes, und der Validationsanfang, also Stringanfang plus Textlänge plus Maskenlänge.

Die zugehörige TEDINFO-Adresse wird also Ob_spec übergeben, damit sind die hier zu erläuternden Varianten fertig. ICONS und IMAGES würden hier die Anfangsadressen der zugehörigen Strukturen haben wollen, das sei - wie gesagt - Ihnen überlassen.

Alle Objektspezifikationen sind nun bekannt, alle Zähler sind auf ihre neuen Werte gesetzt, das Objekt kann in den Baum eingebaut werden.

```
Procedure Define_object
  Gosub Objektdefinitionen
  '
  Repeat
  Until Mousek=1
  Ob_x%=Mousex
  Ob_y%=Mousey
  '
  If Typ%=20 Or Typ%=25 Or Typ%=27
    Gosub Def_obspec(Chr%,Rand%,1,1,Muster%,1)
  Endif
  '
  If Typ%=26 Or Typ%=28
    If Odd(Len(String$))
      String$=String$+Chr$(0)
    Endif
    Bmove Varptr(String$),String_adr%+String_counter%,Len(String$)
    Ob_spec%=String_adr%+String_counter%
    Add String_counter%,Len(String$)
  Endif
  '
  If Typ%=22 Or Typ%=21 Or Typ%=29 Or Typ%=30
    String$=E.mask$+E.text$+E.valid$
    If Odd(Len(String$))
      String$=String$+Chr$(0)
    Endif
    Bmove Varptr(String$),String_adr%+String_counter%,Len(String$)
    Gosub Tedinfo
    Ob_spec%=Tedinfo_adr%+Ted_counter%
```

```
    Add String_counter%,Len(String$)
    Add Ted_counter%,28
  Endif
  Gosub Make_object(Typ%,Flags%,Status%,Ob_spec%,Ob_x%,Ob_y%,50,20)
  Print At(1,21);Space$(63)
Return
'
Procedure Tedinfo
' Font%=3
  Just%=0
  Lpoke Tedinfo_adr%+Ted_counter%,String_adr%+String_counter%
  Lpoke Tedinfo_adr%+Ted_counter%+4,String_adr%+String_counter%+
                                                    Len(E.mask$)
  Lpoke Tedinfo_adr%+Ted_counter%+8,String_adr%+String_counter%+
                                        Len(E.text$)+Len(E.mask$)
  Dpoke Tedinfo_adr%+Ted_counter%+12,Font%
  Dpoke Tedinfo_adr%+Ted_counter%+14,6
  Dpoke Tedinfo_adr%+Ted_counter%+16,Just%
  Dpoke Tedinfo_adr%+Ted_counter%+18,&H1180
  Dpoke Tedinfo_adr%+Ted_counter%+20,0
  Dpoke Tedinfo_adr%+Ted_counter%+22,-1
  Dpoke Tedinfo_adr%+Ted_counter%+24,Len(E.mask$)
  Dpoke Tedinfo_adr%+Ted_counter%+26,Len(E.text$)
  Inc Ted_index%
Return
```

Das so definierte Objekt in seinen Baum einzubauen, heißt, zunächst die Werte in die Objektstruktur einzupoken (Zähler ist hier <Tree_counter%>), dann mit Hilfe einer neuen AES-Funktion <Objc_add> automatisch die Baum-Pointer richtig setzen zu lassen und zum Schluß mit <Objc_draw> das Objekt zu zeichnen.

<Objc_add> wird in einer speziellen Library wohl keinen Sinn machen, denn warum sollte man innerhalb eines Programms einen Objektbaum erweitern wollen? Ich habe deshalb keine Unterprozedur gemacht, sondern die zugehörigen vier Zeilen direkt eingebaut. Die If..Else..Endif-Bedingung sitzt nur deshalb da, weil ich in diese Prozedur auch im Zusammenhang mit dem Wurzelobjekt springe, bei dem nur ein Zeiger zu ändern ist.

```
Procedure Make_object(T.yp%,F.lag%,S.tat%,S.pec%,X.%,Y.%,W.%,H.%)
  Dpoke Tree_counter%+Tree%,-1
  Dpoke Tree_counter%+Tree%+2,-1
  Dpoke Tree_counter%+Tree%+4,-1
  Dpoke Tree_counter%+Tree%+6,T.yp%
  Dpoke Tree_counter%+Tree%+8,F.lag%
  Dpoke Tree_counter%+Tree%+10,S.tat%
  Lpoke Tree_counter%+Tree%+12,S.pec%
  Dpoke Tree_counter%+Tree%+16,X.%
  Dpoke Tree_counter%+Tree%+18,Y.%
  Dpoke Tree_counter%+Tree%+20,W.%
  Dpoke Tree_counter%+Tree%+22,H.%
  If Index%=0
    Dpoke Tree_counter%+Tree%,-1
  Else
    Lpoke Addrin,Tree%          ! Objc_add
    Dpoke Gintin,0
    Dpoke Gintin+2,Index%
    Gemsys 40
  Endif
  Add Tree_counter%,24
  Inc Index%
  Gosub Objc_draw
Return
```

4.13.4 Der Resources-Kopf

Jetzt haben Sie einen wunderschönen Objektbaum errichtet und wollen ihn natürlich innerhalb eines Programms benutzen können - also muß er als Resources-File gesichert werden. Das würde nicht schwerfallen, wenn nur Aufbau und Organisation dieses Files bekannt wären. Die Resources-Library arbeitet ja ständig mit den Adressen der einzelnen Datenstrukturen. Diese müssen also nach Regeln sortiert vorliegen. Es hat mich einiges an Zeit gekostet, Ihnen zumindest die meisten Regeln des RSC-File-Aufbaus zeigen zu können.

Mit den in Word-Format gepackten Daten werden hintereinander die einzelnen Strukturen abgelegt. Angefangen beim Feld <Rs.string>, in dem die ASCII-Daten der Strings und Texte einliegen, über die TEDINFO-Struktur, die ICONBLK-Structur

und die BITBLK-Struktur bis zur eigentlichen Objektstruktur. Existiert eine dieser Strukturen nicht, wird sie einfach übergangen.

Ein Header von exakt 18 Words verweist mit 9 Pointern auf die jeweiligen Anfänge der Einzelstrukturen. Diese Pointer sind keine absoluten Adressen, sondern Offsets auf die Anfangsadresse des Resources-Files, der sich ja wer weiß wo in den freien Speicherraum legen kann. Der Header ist wie folgt aufgebaut:

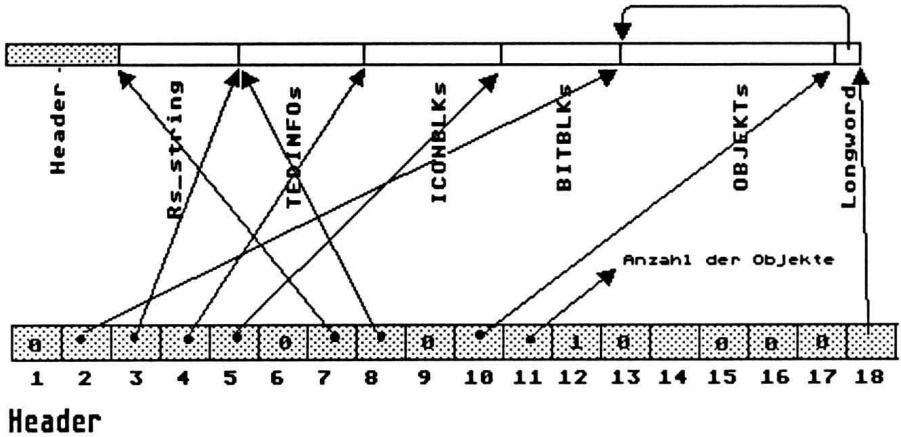


Abb. 29: Headeraufbau

Word 1	immer 0
Word 2	Position des ersten Bytes der Objektstruktur
Word 3	Position des ersten Bytes der TEDINFO-Struktur
Word 4	Position des ersten Bytes der ICONBLK-Struktur
Word 5	Position des ersten Bytes der BITBLK-Struktur

Diese 4 letzten Words haben, falls die entsprechende Struktur nicht vorhanden ist, immer die Adresse der vorherstehenden Struktur. Word 6: wahrscheinlich Position der APPLBLK-Struktur, sonst 0

Word 7	Länge des Headers in Byte, also 36
Word 8	Länge des Headers plus der String- und Textstruktur in Byte
Word 9	immer 0
Word 10	Länge des Files in Byte abzügl. 4
Word 11	Anzahl der Objekte
Word 12	1
Word 13	Anzahl der Tedinfos
Word 14 bis	0. Wahrscheinlich Anzahl von Bitblk,
Word 17	Iconblk usw.
Word 18	Länge des Files in Byte

Am Ende des RSC, nach der letzten Objektdefinition des letzten Objekts folgt ein Longword, das das Anfangsbyte der Objektstruktur aufnimmt, also identisch mit Word 2 des Headers ist. Die Prozedur Rsrc_load arbeitet nun so, daß dieses letzte Longword zu der Anfangsadresse des in den Speicher eingeladenen Files, also HIMEM, addiert wird. Intern kann also wieder mit absoluten Adressen gerechnet werden, die dann nur um die jeweiligen Header-Daten erhöht werden müssen.

4.13.5 Objekte speichern

Mit diesem Wissen kann nun die Prozedur gebaut werden, die Ihre Daten in ein RSC-File umwandelt. Wir haben eine Menge Zeiger auf Adressenbereiche oder Bereichslängen zu verwalten. Sie seien hier noch einmal aufgelistet, um Konfusionen zu vermeiden:

String_pointer%	zeigt auf den Anfang des Rs_string-Bereichs. Da dieser direkt hinter dem Header folgt und dieser eine konstante Länge hat, ist die Größe von String_pointer% = 36.
Stringende_pointer%	zeigt auf das letzte Byte des Rs_string-Bereichs innerhalb des RSC.

String_counter%	zählt die Länge des Rs_string-Bereichs. Diese Variable hat ihre Größe bei der Erstellung der Objektstruktur erhalten.
Ted_pointer%	zeigt auf das erste Byte des Tedinfo-Bereichs innerhalb des RSC.
Ted_counter%	zählt die Länge des Tedinfo-Bereichs. Auch diese Variable hat ihre Größe schon erhalten.
Objc_pointer%	zeigt auf das erste Byte des Object-Bereichs innerhalb des RSC.
Tree_counter%	ist der dazugehörige Längenzähler der Object-Struktur.
Bitblk_pointer% *Bitblk_counter%*	sind die entsprechenden Variablen für den Bitblk-Bereich.
Icon_pointer% *Icon_counter%*	sind die für den ICONBLK-Bereich. Die letzten beiden Strukturen sind im Grundbaukasten nicht vorhanden, sie treten deshalb mit Dummys bzw. gar nicht in Erscheinung.
Rsc_adr%	ist der Pointer auf die Adresse innerhalb des RSC-Files, die gerade bearbeitet wird.
Index%	ist der Objektzähler
Tedinfo_index%	ist der Mengenzähler für die Tedinfos.
Tree%	ist der Anfang des Rsc im RAM.

Zuerst müssen jetzt die einzelnen Pointer mit Werten versehen werden. Etwas verwirrend wird es zunächst, weil wir bisher mit absoluten Adressen gearbeitet haben, da das Resources-File sozusagen lebend erzeugt wurde und GEM natürlich nur diese absoluten RAM-Adressen handhaben kann. Das RSC-File nun arbeitet mit relativen Adressen, also Offsets auf den Anfangsbereich des Files. Derzeit haben wir noch Mischungen beider Adressierungsarten zu verwalten.

Mit den Werten, die bisher bekannt sind, wird der Header nach der oben gezeigten Belegungsliste aufgebaut und an den Anfang des RSC-Speicherbereichs verschoben. Der Zähler Rsc_adr% wird auf 36 gesetzt.

Jetzt nehmen wir unsere Baumadresse und wandern Objekt für Objekt durch das innerhalb des RAM liegende RSC, denn auch hier müssen wir absolute in relative Adressen ändern: Überall da, wo Ob_spec auf eine Strukturadresse zeigt, wird diese relativiert. Außerdem sind die Objektkoordinaten innerhalb eines RSC anders abgelegt als im RAM: Im unteren Byte liegt die Koordinate, die sich auf Buchstaben-Scanlines bezieht, es können also nur Werte im 8*16 Raster auftreten. Im oberen Byte, als Offset auf diese Koordinatenwerte, wird die Position differenziert. Beim Laden des RSC-Files werden die Koordinatenwerte dann umgerechnet. Ich habe hier der Einfachheit halber auf diese Differenzierung verzichtet und nur die Koordinatenwerte in das 8*16-Raster überführt.

Ebenfalls innerhalb der Tedinfo-Struktur werden Adressen von Absolut- in Relativwerte umgerechnet, nämlich jeweils in den ersten drei Longwords.

So, nun kann unser RSC weiter aufgebaut werden: Falls vorhanden, wird der Rs_string_Bereich in den RSC-Bereich verschoben und der Rsc_adr%-Zähler entsprechend erhöht. Weiter wird, falls vorhanden, der Tedinfo-Bereich darangehängt und der Rsc_adr%-Zähler wieder erhöht. Dasselbe könnte jetzt mit den BITBLK- und ICONBLK-Strukturen erfolgen.

Zum Schluß der Speicherraumbewegungen wird der Object-Bereich an das Ganze gehängt und ganz ans Ende das Longword für den Pointer auf den Anfang der OBJECT-Struktur innerhalb des RSC eingepoked.

Zwei Angaben für den Header wissen wir erst jetzt: nämlich die Gesamtlänge des Files und die Gesamtlänge abzüglich 4 für das 18- und 34-Word. Diese werden nachträglich in den Header eingetragen, und das RSC kann mit Hilfe des <Bsave>-Befehls auf Diskette geschrieben werden.

```
Prozedur Save_rsc
  String_pointer%=36
  Stringende_pointer%=String_pointer%+String_counter%
  If Ted_counter%>0
    Ted_pointer%=String_pointer%+String_counter%
    Objc_pointer%=Ted_pointer%+Ted_counter%
  Else
    Objc_pointer%=String_pointer%+String_counter%
    Ted_pointer%=Objc_pointer%
  Endif
  Icon_pointer%=Ted_pointer%
  Bitblk_pointer%=Ted_pointer%
  '
  Header$=Mki$(0)+Mki$(Objc_pointer%)+Mki$(Ted_pointer%)+
          Mki$(Icon_pointer%)+Mki$(Bitblk_pointer%)+Mki$(0)+
          Mki$(String_pointer%)+Mki$(Stringende_pointer%)+Mki$(0)
  Header$=Header$+Mki$(0)+Mki$(Index%)+Mki$(1)+Mki$(Ted_index%)+
                          Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)
  Bmove Varptr(Header$),Rsc_adr%,36
  Add Rsc_adr%,36
  '
  A%=Tree%
  For N%=0 To Index%-1
    If Dpeek(A%+6)=26 Or Dpeek(A%+6)=28
      Print Lpeek(A%+12)'String_adr%
      Lpoke A%+12,Lpeek(A%+12)-String_adr%+36
      Print Lpeek(A%+12)
    Endif
    If Dpeek(A%+6)=21 Or Dpeek(A%+6)=22 Or Dpeek(A%+6)=29
                      Or Dpeek(A%+6)=30
      Lpoke A%+12,Lpeek(A%+12)-Tedinfo_adr%+Stringende_pointer%
      Print Lpeek(A%+12)
    Endif
    Dpoke A%+16,Int(Dpeek(A%+16)/8)
    Dpoke A%+18,Int(Dpeek(A%+18)/16)
    Dpoke A%+20,Int(Dpeek(A%+20)/8)
    Dpoke A%+22,Int(Dpeek(A%+22)/16)
    Add A%,24
  Next N%
  For N%=Tedinfo_adr% To Tedinfo_adr%+Ted_counter% Step 28
    Lpoke N%,Lpeek(N%)-String_adr%+36
    Lpoke N%+4,Lpeek(N%+4)-String_adr%+36
    Lpoke N%+8,Lpeek(N%+8)-String_adr%+36
  Next N%
  '
```

```
  If String_counter%>0
    Bmove String_adr%,Rsc_adr%,String_counter%
    Add Rsc_adr%,String_counter%
  Endif
  If Ted_counter%>0
    Bmove Tedinfo_adr%,Rsc_adr%,Ted_counter%
    Add Rsc_adr%,Ted_counter%
  Endif
  Bmove Tree%,Rsc_adr%,Tree_counter%
  Add Rsc_adr%,Tree_counter%
  Lpoke Rsc_adr%,Objc_pointer%
  Add Rsc_adr%,4
  Dpoke Varptr(Rsc$)+18,Rsc_adr%-Varptr(Rsc$)-4
  Dpoke Varptr(Rsc$)+34,Rsc_adr%-Varptr(Rsc$)
  '
  Bsave "TEST.RSC",Varptr(Rsc$),Rsc_adr%-Varptr(Rsc$)
Return
```

Dieses so erzeugte RSC kann mittels derselben Methoden, die in den ersten Kapiteln beschrieben wurden, in Ihr Programm eingelagert und bearbeitet werden.

5. Spezialitäten aus Kraut und Rüben

5.1 FORM_INPUT, etwas anders

Der Befehl <Form_input> in GFA-BASIC, so mächtig er ist, hat für manche Anwendungen einen gravierenden Nachteil: Beenden ist nur mittels der RETURN-Taste möglich. In vielen Fällen will man jedoch mit Pfeiltasten oder HOME aussteigen, will nach Erreichen der maximalen Maskenlänge automatisch in Subroutinen springen können oder hat sonstige Ideen im Kopf, die durch das Betätigen von RETURN behindert würden.

Für diese Fälle folgt hier eine alternative Form_input-Routine. Sie schickt ein Flag zurück, das folgende Werte haben kann:

1. nach Erreichen der maximalen Eingabelänge
2. nach Betätigen der RETURN-Taste
3. nach Drücken der "Cursor runter"-Taste
4. nach Drücken der "Cursor hoch"-Taste
5 nach Drücken von Clr/Home

Außerdem kann mit der Insert-Taste vom Überschreib- auf den Einfügemodus umgeschaltet werden.

Diese Procedure ist enorm vielseitig. Mit ein bißchen Bastelei kann sie auch zur kompletten Textverarbeitungsroutine erweitert werden. Deshalb hier eine etwas ausführlichere Erläuterung.

Innerhalb einer Do..loop-Schleife wird Inp(2), also die Eingabe von Tastatur auf den Bildschirm abgefragt. Inp(2) gibt für alle ASCII-Zeichen deren ASCII-Wert zurück, für alle Sondertasten einen Atari-spezifischen Wert. Ausgehend von diesem Wert wird in Unterprozeduren verzweigt, wenn:

- zwar ein ASCII-Zeichen, aber kein druckbares Zeichen eingegeben wurde. Das sind hier: Backspace (8), Delete (127), Return (13), Tab (9) und ESC (27).
- ein Sonderzeichen außerhalb des normalen ASCII-Codes eingegeben wurde.

Ansonsten wird der String X$ zusammengesetzt, indem aus der momentan aktuellen Spaltenposition heraus der String geteilt und das neue Zeichen an die Spaltenposition geschrieben wird. Entweder als Ersatz des Zeichens auf der Spaltenposition (Überschreiben) oder, indem der gesamte Stringrest um eine Position nach rechts verschoben wird (Einfügen).

Anschließend wird der gesamte String an seine Anfangskoordinaten geschrieben.

Da GEM in Fenstern - und das wird wohl die häufigste Anwendung sein - keinen Cursor anbietet, muß dieser selbstgestrickt werden. Am einfachsten, denke ich, ist der hier beschrittene Weg, der ein Sprite definiert und dieses jeweils koordinatengerecht schreibt. Die Koordinatenberechnerei sowohl bei der <Text..>- als auch bei der <Sprite..>-Zeile hat u.a. damit zu tun, daß der Grafikursprung für Sprites immer, auch im geöffneten Fenster, bei Bildschirm 0/0 liegt, der Text-Befehl hat seine Ursprungskoordinaten im Fenster jedoch bei 0/19. Ich habe in den beiden Zeilen, in denen das Sprite geschrieben wird, jeweils am Ende der Vertikalkoordinate eine 19 stehen. Diese gilt für Fenster, bei sonstigen Anwendungen ist diese 19 zu streichen.

Die <Procedure No_charakter> setzt den String für DELETE und Backspace unterschiedlich zusammen: Einmal wird das Zeichen unter dem Cursor gelöscht, dieser bleibt an seiner alten Stelle stehen und der rechte Stringteil wird herangezogen. Das andere Mal wandert der Cursor nach links, löscht das jetzt unter ihm liegende Zeichen und zieht dann erst den Reststring heran. Die RETURN-Taste belegt lediglich das Flag Ret% mit 1, was einen Ausstieg aus Form_Input-Routine bewirkt. TAB und ESC sind nicht definiert.

Die <Procedure No_ascii> hantiert hier ausschließlich den Cursortastenblock, obwohl sie natürlich noch viel mehr könnte (so kann bei Bedarf jede einzelen Funktionstaste belegt werden). Innerhalb der Routine - weil wohl immer auf diese Art funktionierend - wird der Spaltenzähler verändert. Da der Zeilenzähler außerhalb der <Form_input>-Routine verändert werden muß, wird das Flag Ret% mit 3 bzw. 4 bzw. 5 belegt. Insert schaltet wiederum ein Flag um.

```
! ******************************************************************
! * Maskeneingabeprozedur. Globaler Rückgabewert: Ret%. Dieser kann *
! * die Werte 1 bis 5 haben. Siehe Programmkommentare              *
! * Vor der Erstbenutzung: <Gosub Spritedefs> eingeben             *
! ******************************************************************
! Programm: FORM_INP.BAS
!                                          ! Übergabeparameter:
Procedure Form_input(Ptr%,X$,Sp%,Ze%,Laenge%) ! Ptr%=Rückgabestring
  Local Y%                                ! X$=Stringvorbelegung
  Ret%=0                                  ! Sp%=Anfangsspalte
  Y%=Len(X$)+1                            ! Ze%=Anfangszeile
  Text Sp%*8-8,Ze%*16,X$+" "              ! Laenge%=maximale Länge
  Sprite Cur$,(Sp%+Y%-1)*8,Ze%*16-2+19
  Do
    T%=Inp(2)
    !
    If T%>=187
      Gosub No_ascii_taste(T%)
    Else
      If T%=27 Or T%=8 Or T%=127 Or T%=13 Or T%=9
        Gosub No_character(T%)
      Else
        If Insert!
          X$=Left$(X$,Y%-1)+Chr$(T%)+Mid$(X$,Y%)
        Else
          X$=Left$(X$,Y%-1)+Chr$(T%)+Mid$(X$,Y%+1)
        Endif
        Inc Y%
        If Y%=Laenge%                         ! Ret% =1
          Ret%=1
        Endif
      Endif
    Endif
    Sprite Cur$
    Text Sp%*8-8,Ze%*16,X$+" "
```

```
    Sprite Cur$,(Sp%+Y%-1)*8,Ze%*16-2+19
    *Ptr%=X$
    Exit If Ret%>0
  Loop
Return
'
Procedure No_character(Sign%)                    !
  If Sign%=8 And Y%>1                            ! Backspace
    Dec Y%                                       !
    X$=Left$(X$,Y%-1)+Mid$(X$,Y%+1)              !
  Endif                                          !
  If Sign%=13                                    ! CR/LF
    Ret%=2                                       ! dann Ret%=2
  Endif                                          !
  If Sign%=127 And Y%>1                          ! Delete
    X$=Left$(X$,Y%-1)+Mid$(X$,Y%+1)              !
  Endif                                          !
Return                                           !
'                                                !
Procedure No_ascii_taste(Scan%)                  !
  If Scan%=210                                   ! Inserttaste
    If Insert!=0                                 !
      Insert!=-1                                 !
    Else                                         !
      Insert!=0                                  !
    Endif                                        !
  Endif                                          !
  '                                              !
  If Scan%=199                                   ! CLR-Home
    Ret%=5                                       ! Cursor an Anfang
  Endif                                          ! ret%=5
  If Scan%=203 And Y%>1                          ! Cursor links
    Dec Y%
  Endif                                          !
  If Scan%=205 And Y%<=Laenge%                   ! Cursor rechts
    Inc Y%                                       !
  Endif                                          !
  If Scan%=208                                   ! Cursor nach unten
    Ret%=3                                       ! Ret%=3
  Endif                                          !
  If Scan%=200                                   ! Cursor nach oben
    Ret%=4                                       ! Ret%=4
  Endif                                          !
Return                                           !
'
```

```
Procedure Spritedefs
  Cur$=Mki$(15)+Mki$(12)+Mki$(1)+Mki$(0)+Mki$(1)+Mki$(0)+Mki$(0)+Mki$(0)
+Mki$(0)+Mki$(0)
  Cur$=Cur$+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$
(0)+Mki$(0)+Mki$(0)+Mki$(0)
  Cur$=Cur$+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$
(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(0)+Mki$(511)+Mki$(0)+Mki$(511)
Return
```

Was kann man mit dieser Routine machen? Ich zeige Ihnen ein Beispiel für Maskenprozeduren. Sie definieren - ob in Data-Zeilen oder anders, ist egal - Ihre Maskenpositionen und laufen in einer Schleife die Form_input-Routine immer wieder an. RETURN oder Cursor-down lassen das nächste Maskenfeld zur Bearbeitung zu, Cursor-up das vorherige und Home das erste.

```
Dim String$(3),S%(3,3)
Openw 0
Gosub Spritedefs
'
Data "",10,10,25
Data "",10,11,30
Data "",12,12,20
For N%=1 To 3
  Read String$(N%),S%(N%,1),S%(N%,2),S%(N%,3)
Next N%
N%=1
Repeat
  Gosub Form_input(*String$,String$(N%),S%(N%,1),S%(N%,2),S%(N%,3))
  String$(N%)=String$
  If Ret%=1 or Ret%=2 Or Ret%=3
    Inc N%
  Else
    If Ret%=5
      N%=1
    Else
      If Ret%=4
        Dec N%
          If N%<1
          N%=1
          Endif
```

```
      Endif
    Endif
  Endif
Until N%=4
```

Diese Routine zeigt Ihnen die generellen Arbeitshinweise zur Formular-Erstellung. An einigen Stellen kann diese Routine von Ihnen sicher noch verbessert werden. Ich denke dabei an die beschränkte Cursorbewegung innerhalb eines Wortes und an die Möglichkeit, z.B. mit <ESC> den ganzen String zu löschen.

5.2 Automatischer Zeilenumbruch für Textverarbeitung

Manchmal ist es ganz schön, auch in Formularmasken die Möglichkeit zu haben, automatisch beim Erreichen des Zeilenendes in die nächste Zeile zu springen und dabei das "angebrochene" Wort aus dem vorherigen String zu entfernen und als erstes Wort des nächsten zu benutzen: also das tun zu können, was jede bessere Textverarbeitung können muß.

```
' Programm: WORDWRAP.BAS
'
Procedure Word_wrap(A.%,X.$,P.tr%)
  S.p%=Len(X.$)
  While S.p%>1
    Exit If Mid$(X.$,S.p%,1)=" "
    T$=Mid$(X.$,S.p%)
    Print " ";Chr$(8);Chr$(8);
    Dec S.p%
  Wend
  '
  X.$=Left$(X.$,S.p%-1)
  T$=T$+Chr$(A.%)
  *P.tr%=T$
Return
```

Hinein kommt das letzte eingegebene Zeichen und der letzte Gesamtstring, zurückgegeben wird der neue String, der nur das Teilwort mit dem zuletzt eingegebenen Zeichen enthält. Ansonsten ist außer der Kürze an dieser Prozedur nichts Bemerkens-

wertes: Der Altstring wird zurückgelesen, bis ein Leerzeichen auftritt. Gleichzeitig wird der gelesene Teil der Zeile und des Strings gelöscht. Der gelöschte Teil wird in einen neuen String überführt und als neuer String ausgegeben.

Jetzt können Sie aus der Form_input-Prozedur bei der Bedingung "maximale Eingabelänge erreicht" in die Word_wrap-Procedure springen und haben schon fast so etwas wie Textverarbeitung zusammen.

5.3 Druckeranpassung statt Deskaccessory

Wenn Sie sich auch schon darüber geärgert haben, daß Sie zwar einerseits einen Drucker haben, der Hardcopies in 960er Breite druckt, und Sie deshalb mit der Betriebssystemvoreinstellung nichts anfangen können, andererseits aber nur deswegen zwei Deskaccessory-Plätze verschwenden müssen, dann habe ich hier etwas für Sie.

Wissen Sie, daß DESKTOP.INF auch die Voreinstellungen aus der Druckeranpassungs und übrigens auch aus der Anpassung der seriellen Schnittstelle auf Diskette sichert? Wenn nicht, probieren Sie es aus: Irgend etwas anklicken, ARBEIT SICHERN aus dem Menü anklicken, neu booten, und Ihre Einstellung erscheint wieder in Deskaccessory.

Das kann simuliert werden. Die 10 Zeilen des Programms lesen die zuständige Information aus DESKTOP.INF heraus, wandeln sie in XBIOS lesbare Daten um und übergeben diese dann der zuständigen XBIOS-Routine (XBIOS 33). Das funktioniert natürlich am allerbesten, wenn Sie den Source compilieren und in einen AUTO-Ordner stecken. Aber auch eingebaut in ein selbstgebautes Programm sieht die Routine nicht übel aus.

```
' Programm: DESKINF.BAS
'
Open "I",#1,"DESKTOP.INF"
Line Input #1,A$
Line Input #1,A$
```

```
For N%=5 Downto 0
  X$=Mid$(A$,N%+3,1)
  If X$="1"
    A%=A%+2^N%
  Endif
Next N%
Void Xbios(33,A%)
```

Dazu zwei Erläuterungen: Im DESKTOP.INF ist die zweite Zeile für die Druckeranpassung und die erste für die RS232-Schnittstelle zuständig. Dabei gilt:

RS232

Hinter dem Deklarationszeichen #a folgen 6 Zahlen:

1 0 = Vollduplex, 1 = Halbduplex
2 Baudrate: 0=9600, 1=4800, 2=1200, 3=300
3 Parity: 0=None, 1=odd, 2=even
4 Bits/Zeichen: 0=8, 1=7, 2=6, 3=5
5 Protokoll: 1=XON/XOFF, 0=RTS/CTS
6 Bit 8: 0 = gesetzt, 1 = nicht gesetzt

Druckeranpassung

Hinter dem Deklarationszeichen #b folgen 6 Zahlen:

1 Druckertyp: 0=Matrix, 1=Typenrad
2 Farbe: 0=Schwarz/weiß, 1=Farbe
3 Zeilenlänge: 0=1280 Zeichen, 1=960 Zeichen/Zeile
4 Quality: 0=Draft, 1=NLQ
5 Druckerausgang: 0=Centronics, 1=RS232
6 Papierart: 0=endlos, 1=Einzelblatt

Der XBIOS-Routine 33, Setprt, ändert die Druckerkonfiguration. Im Ein-Byte-Parameter haben die Bits folgende Bedeutung:

0 0=Matrixdrucker, 1= Typenraddrucker
1 0=schwar/weiß, 1=Farbe
2 0=1280 Zeichen, 1=960 Zeichen
3 0=Draft, 1=NLQ

4	0=Centronics, 1=RS232
5	0=Endlospapier, 1=Einzelblatt
6-14	reserviert
15	muß 0 sein

Sie sehen, die Forderungen dieser Routine nach Bitbelegung werden exakt von der Zahlenbelegung des DESKTOP.INF erfüllt. Wir können also dieses direkt als Bitmuster einlesen. Auf fast identische Art kann die XBIOS(15)-Routine Rsconf mit den Daten aus DESKTOP.INF gefüllt werden, um die Schnittstellenparameter der RS232-Schnittstelle einzustellen.

5.4 Uhrzeit aus dem AUTO-Ordner

Hier ist noch etwas für Ihren AUTO-Ordner. Dazu aber erst einmal folgendes: Der Atari fragt den AUTO-Ordner fast am Anfang seiner Boot-Prozedur ab. GEM ist noch nicht geladen, wenn AUTO-Programme ausgeführt werden. Das hat manchmal fatale Folgen: Wenn Sie nämlich ein Programm in den AUTO-Ordner stecken, das auch nur eine einzige VDI- oder AES-Routine ausführen muß, stürzt der Rechner ab. Das heißt: keine Line-, Draw- oder Plot-Befehle hinein, kein Fenster oder Menü, nur das, was auch in GFA-BASIC über BIOS und XBIOS-Routinen läuft. Traurige Konsequenz ist, daß selbststartende Programme fast unmöglich sind. Schade, aber so ist es nun mal.

Deshalb hier eine Routine, die - compiliert - in den AUTO-Ordner paßt und Ihnen ermöglicht, vor Startbeginn Zeit und Datum einzugeben, so daß Sie immer gezwungenermaßen auf dem aktuellen Stand sind. Bis auf eine Ausnahme ist das Programm banal. Das Systemdatum wird auf den Bildschirm geschrieben und zu einer Eingabeprozedur gesprungen. Nach Eingabe von RETURN geht's gleich weiter (falls Sie das Datum nicht interessiert). Dasselbe für die Zeitangaben: Abfrage nach korrekten Eingaben, wenn nicht, das Ganze nochmal, ansonsten Ende der Bearbeitung.

```
' Programm: TIME.BAS
'
If Mid$(Time$,5,1)<"1"
  Anfang:
  Print At(30,10);Date$'''Time$
  Print At(30,10);Chr$(27);"e";
  Gosub Eingabe(31,30)
  If A$<>Chr$(13)
    D$=A$+"."
    Gosub Eingabe(12,33)
    D$=D$+A$+".1987"
  Endif
  Gosub Eingabe(23,43)
  If A$<>Chr$(13)
    T$=A$+":"
    Gosub Eingabe(59,46)
    T$=T$+A$+":00"
  Endif
  Print At(55,10);"OK? ";
  Do
    Ant$=Inkey$
    Exit If Ant$<>""
  Loop
  Print Ant$;
  If Ant$="n" Or Ant$="N"
    Print At(55,10);"     "
    Goto Anfang
  Endif
  Print Chr$(27);"f";
  Settime T$,D$
Endif
Quit
```

Ich finde die <Procedure Eingabe> ganz interessant. Es handelt sich hier nämlich um eine rekursive Prozedur. Das heißt, sie ruft sich selbst auf. Sowohl in der Zeit als auch im Datum muß ich ja immer 2 Ziffern hintereinander eingeben, diese zu einer Zahl zusammenfassen und dann eine Cursorposition weiterspringen. Alle Zahlen, die einzugeben sind, haben ihren Grenzwert: Das Tagesdatum kann z.B. nicht höher als 31 sein, das Monatsdatum nicht größer als 12, die Minuten nicht mehr als 59 usw. Eingabefehler möchte ich vermeiden. Ich mache jetzt folgendes: Mittels des bekannten Inkey$ wird ein Tastendruck in einen String übernommen. Ob die eingegebene Taste überhaupt eine Zahl ist,

wird durch den Instr(...)-Befehl abgefragt. Falls die Zahl außerhalb des Grenzwertes gerät, wird einfach die Prozedur noch einmal von vorne angefangen. Die Übergabeparameter werden dabei genauso wieder mitübergeben. Profis werden jetzt sagen, was denn, das ist doch nur sehr, sehr bedingt rekursiv. Na ja, sicher, aber erläutert es nicht die Möglichkeiten, ohne gleich in höchste Mathematik einsteigen zu müssen?

```
Procedure Eingabe(Lim%,X%)
  A$=""
  Print At(X%,10);
  Do
    X$=Inkey$
    If X$<>""
      If Instr("1234567890",x$)
        A$=A$+X$
        Print X$;
        If Val(A$)>Lim%
          Gosub Eingabe(Lim%,X%)
        Endif
      Endif
    Endif
    Exit If Len(A$)=2 Or A$=Chr$(13)
  Loop
Return
```

5.4.1 Die VT52-Bildschirmsteuerung

Oben habe ich einige Print-Befehle benutzt, die das ESC (Chr$(27)), gefolgt von einem Buchstaben schreiben. Dabei ist eine auf TOS-Ebene wunderschöne Implementierung des Atari ausgenutzt: die VT52-Escapes. Der VT52-Emulator hat nämlich eine komplette Cursorsteuerung eingebaut, die aber - wie gesagt - leider nicht mehr in Fenstern funktioniert. Aber dennoch: Nicht alles, was zu programmieren ist, braucht diese Fenster. Deshalb eine kurze Liste der Möglichkeiten:

Alle Cursor-Befehle lauten gleichermaßen <Print chr$(27)+"..."; Das Semikolon am Ende ist wichtig, sonst haben Sie wenig von dem Aufwand. Im folgenden liste ich nur den Buchstaben auf, der in die Anführungszeichen kommt:

e Cursor an

f Cursor aus

A Cursor up

B Cursor down

C Cursor right

D Cursor left

E Clear/Home (Bildschirm löschen und Cursor auf 0/0)

H Cursor Home ohne Löschen des Bildschirms

I Scroll down (Cursor hoch. Befindet sich der Cursor am oberen Bildschirmrand, Leerzeile einfügen und den Restbildschirm nach unten scrollen.

J Bildschirm von Cursor bis zum Ende löschen

K Bildschirm von Cursor bis zum Zeilenende löschen

L Leerzeile einfügen und Bildschirmrest nach unten scrollen

M Zeile löschen und den Bildschirmrest nachrücken.

Y Cursor auf beliebige Bildschirmposition setzen. Zwei weitere Parameter folgen auf das "Y", die die Zeile und die Spalte angeben. Beide Werte müssen mit einem Offset von 32 versehen werden. (Zeile 5, Spalte 12 hieße also: Print chr$(27)+"Y"+37+44

b Farbe der Buchstaben

c Farbe des Hintergrundes

d Löschen des Bildschirms von Anfang bis zur Cursorposition

l Löscht die Cursorzeile, ohne den Restbildschirm nachzuziehen

o Löscht vom Zeilenanfang bis zur Cursorposition

p Reverse Darstellung aller nachfolgenden Eingaben

q schaltet den Reversmodus wieder aus

v Schaltet Zeilenüberlauf an

w Schaltet Zeilenüberlauf aus

j Speichert die aktuelle Cursorposition

k Setzt den Cursor auf die mit ESC j gespeicherte Position zurück.

5.5 Und noch 'ne Uhr

In manchen Programmen ist es ganz witzig, auf Knopfdruck die Uhrzeit lesen zu können. Machen wir's doch einmal anders, als oben in der Statuszeile und sagen, wenn der Benutzer auf z.B. die ESC-Taste drückt, erscheint eine Uhr auf dem Bildschirm, aber eine richtige runde Uhr mit Zeigern und Ziffernblatt.

```
' Programm: UHR.BAS
'
Do                      ! Dies sei das Anwenderprogramm, in dem
  If Inkey$=Chr$(27)    ! innerhalb einer Schleife die Abfrage-
    Gosub Uhr           ! bedingung steht. Statt ESC kann natürlich
  Endif                 ! jede andere Taste genommen werden.
Loop
```

Die Uhr selber bedarf etwas Geometrie, mehr nicht. M_x% und M_y% geben den Mittelpunkt der Uhr vor, Size% ihre Größe. Diese Variablen lassen sich leicht von Ihnen ändern.

Die Stunden"zahlen" werden als Striche im Uhrzeigersinn um den Mittelpunkt konzentriert gezeichnet. In einer Endlosschleife mit irgendeiner Ausstiegsbedingung wird die Zeit aus der Tastaturprozessoruhr abgefragt und in ihre Bestandteile Sekunden, Minuten, Stunden zerlegt. Das Zeichnen der drei Zeiger geschieht im Graphmode 3. Der Timer-Befehl gibt die einigermaßen genauen Zeitvorgaben zum Neuzeichnen der Uhr einmal alle zwei Sekunden, da die interne Atari-Uhr auch nur in diesem Takt läuft.

```
Procedure Uhr
  Local M_x%,M_y%,Size%,Uhrx,Uhry,Sek,Min,Std,T%
  M_x%=580
  M_y%=70
  Size%=50
  Box M_x%-Size%,M_y%-Size%,M_x%+Size%,M_y%+Size%
  For I=1 To 12
    Uhrx=Sin(2*Pi*I/12)
    Uhry=Cos(2*Pi*I/12)
    Line M_x%+(Size%-7)*Uhrx,M_y%+(Size%-7)*Uhry,
                        M_x%+(Size%-2)*Uhrx,M_y%+(Size%-2)*Uhry
  Next I
  Graphmode 3
  Repeat
    Sek=Val(Mid$(Time$,7))*Pi/30
    Min=Val(Mid$(Time$,4))*Pi/30+Sek/60
    Std=Val(Time$)*Pi/6+Min/12
    Defline 1,3
    Line M_x%,M_y%,M_x%+Size%/2*Sin(Std),M_y%-Size%/2*Cos(Std)
    Line M_x%,M_y%,M_x%+(Size%-Size%/10)*Sin(Min),
                                M_y%-(Size%-Size%/10)*Cos(Min)
    Defline 1,1
    Line M_x%,M_y%,M_x%+Size%*Sin(Sek),M_y%-Size%*Cos(Sek)
    T%=Timer
    Repeat
    Until T%+400=Timer
    Defline 1,3
    Line M_x%,M_y%,M_x%+Size%/2*Sin(Std),M_y%-Size%/2*Cos(Std)
    Line M_x%,M_y%,M_x%+(Size%-Size%/10)*Sin(Min),
                              M_y%-(Size%-Size%/10)*Cos(Min)
    Defline 1,1
    Line M_x%,M_y%,M_x%+Size%*Sin(Sek),M_y%-Size%*Cos(Sek)
  Until Mousek=2 Or Inkey$<>""
  Cls
  Graphmode 1
Return
```

5.6 Ein bißchen Perspektive gefällig

Tips & Tricks in einem solchen Buch verlangen auch ihre Erläuterung. Ich hoffe, diesem Anspruch sind wir Verfasser gerecht geworden. Ich weiche trotz alledem jetzt von diesem Prinzip ab und gebe Ihnen 3 Prozeduren an die Hand, ohne die Hintergründe zu erklären. Es handelt sich um 2 Prozeduren, die

Körper definieren, und um die dazugehörige Zeichenprozedur. Ich sage Ihnen auch, warum: Es handelt sich um ausgewachsene Perspektivdarstellungen, ja, sogar erweitert um einen einfachen Hidden_line_Algorithmus. Und das gehört in die höhere Mathematik, die - wenn der Erklärungsanspruch befriedigt werden sollte - eines ganzen Buches bedarf. Ich habe es probiert, die Sache zu beschreiben, die hier passiert. Nach 50 Seiten trockenster Materie bin ich ausgestiegen. Erlauben Sie auch einem Autor, irgendwann die Lust zu verlieren.

Deshalb hier der Quellcode in Kurzkommentierung. Die Prozedur finden Sie unter dem Namen "3DGRAFIK.BAS" auf der Diskette. Nach dem Starten wird eine Balkengrafik erscheinen. Für die Kuchengrafik entfernen Sie bitte die Rem-Zeilen vor dem Aufrufen des Unterprogramms Def-torte und setzen die Def-Quader-Aufrufe in Rem-Zeilen.

Ziel der Arbeit war, für Präsentationsgrafiken Diagramme zeichnen zu können, die einem ästhetisch höheren Anspruch genügen. Es sind zwei verschiedene Diagrammformen herausgekommen: das Balkendiagramm, das durch eine Aufeinanderschichtung und Reihung von einzelnen Kuben erzeugt wird, und das Tortendiagramm, dessen Ursprung eine segmentierte Säule ist, deren einzelne Segmente die Gesamttorte ergeben. Einzelne Stücke daraus kann man herausziehen.

Der komplette Algorithmus ist eine Kombination aus punkt- und flächenorientierten Bestimmungen. Es sind zunächst für die mögliche Anzahl der Eckpunkte in allen drei Koordinaten Arrays zu dimensionieren. Diese werden später umgerechnet in Bildschirmkoordinaten. Auch diese müssen dimensioniert sein. Für die Flächen, deren Anzahl natürlich geringer ist als die der Eckpunkte, werden 4 Arrays benötigt: Der Anfangseckpunkt der Fläche, die Anzahl der Punkte pro Fläche, die Farbe, die die Fläche auf der Zeichnung haben soll, und die relative Entferung der Fläche vom Ursprung.

```
Dim X(1000),Y(1000),Z(1000)
Dim Xx%(1000),Yy%(1000)
Dim X_monitor%(1000),Y_monitor%(1000),Pkt_entf(1000)
Dim Start%(200),Anz%(200),Farbe%(200),Entfernung(200)
```

5.6.1 Der Kubus

In der ersten Abteilung wird nun ein Quader oder Würfel definiert. Übergeben werden müssen seine drei Koordinaten, sowie die Breite, Höhe und Farbe, in der er dargestellt werden soll.

Nun wird jede Fläche, die der Quader hat, definiert. Sein Anfangspunkt, die Bildschirmkoordinaten der einzelnen Eckpunkte und die Farbe wird in das jeweils zugehörige Array eingerechnet. Sie sehen: Ab jetzt geht's wirklich mit Flächen und nicht mit Begrenzungslinien weiter. Das ist - auch von der Zeichnungssystematik her - ein großer Unterschied zu den meisten sogenannten 3-D-Programmen, die nur linienorientiert arbeiten und daher weder überdeckte Linien verstecken, noch Farbinformationen für einzelne Körperbereiche unterschiedlich bestimmen können. Hier z.B. wird für die Deckfläche des Kubus die übergebende Farbe um eins erhöht, also etwas dunkler gemacht, die Seitenflächen erhalten die übergebene Farbe und die Vorder- und Rückfläche werden sehr viel dunkler angesetzt.

```
Procedure Def_quader(X%,Y%,Z%,Breite%,Hoehe%,Farbe%)
  Inc Fl%
  Start%(Fl%)=Pkt%+1
  Anz%(Fl%)=4
  Farbe%(Fl%)=Farbe%+1
  '
  Inc Pkt%
  X(Pkt%)=X%
  Y(Pkt%)=Y%
  Z(Pkt%)=Z%+Hoehe%
  Inc Pkt%
  X(Pkt%)=X%+Breite%
  Y(Pkt%)=Y%
  Z(Pkt%)=Z%+Hoehe%
  Inc Pkt%
  X(Pkt%)=X%+Breite%
  Y(Pkt%)=Y%+Breite%
  Z(Pkt%)=Z%+Hoehe%
  Inc Pkt%
  X(Pkt%)=X%
  Y(Pkt%)=Y%+Breite%
  Z(Pkt%)=Z%+Hoehe%
  ' ----- links und rechts
```

```
For I%=X% To X%+Breite% Step Breite%
  Inc Fl%
  Start%(Fl%)=Pkt%+1
  Anz%(Fl%)=4
  Farbe%(Fl%)=Farbe%
  '
  Inc Pkt%
  X(Pkt%)=I%
  Y(Pkt%)=Y%
  Z(Pkt%)=Z%
  Inc Pkt%
  X(Pkt%)=I%
  Y(Pkt%)=Y%+Breite%
  Z(Pkt%)=Z%
  Inc Pkt%
  X(Pkt%)=I%
  Y(Pkt%)=Y%+Breite%
  Z(Pkt%)=Z%+Hoehe%
  Inc Pkt%
  X(Pkt%)=I%
  Y(Pkt%)=Y%
  Z(Pkt%)=Z%+Hoehe%
Next I%
' ----- vorne und hinten
For I%=Y% To Y%+Breite% Step Breite%
  Inc Fl%
  Start%(Fl%)=Pkt%+1
  Anz%(Fl%)=4
  Farbe%(Fl%)=Farbe%+2
  '
  Inc Pkt%
  X(Pkt%)=X%
  Y(Pkt%)=I%
  Z(Pkt%)=Z%
  Inc Pkt%
  X(Pkt%)=X%+Breite%
  Y(Pkt%)=I%
  Z(Pkt%)=Z%
  Inc Pkt%
  X(Pkt%)=X%+Breite%
  Y(Pkt%)=I%
  Z(Pkt%)=Z%+Hoehe%
  Inc Pkt%
  X(Pkt%)=X%
```

```
        Y(Pkt%)=I%
        Z(Pkt%)=Z%+Hoehe%
      Next I%
    Return
```

--

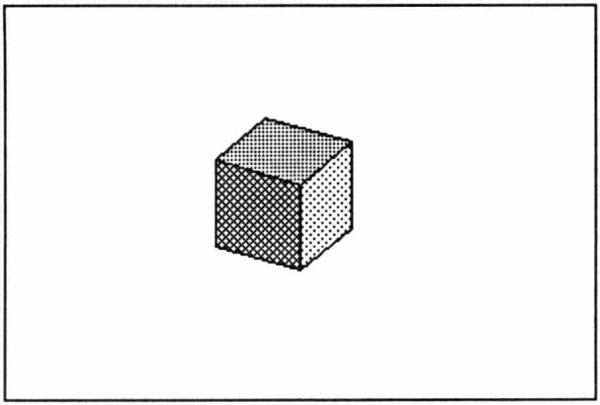


Abb. 30: Quader

5.6.2 Das Tortenstück

Das Tortenstück will die Koordinaten, den Radius, die Höhe, den Anfangswinkel relativ zu einer vertikalen Linie durch den Tortenmittelpunkt, den dazugehörigen Endwinkel und ebenfalls die Farbe übergeben haben.

Es werden dann genau wie beim Quader Flächen definiert, nur daß es hier natürlich mehr Randflächen sind, damit der runde Eindruck entstehen kann.

```
Procedure Def_torte(X%,Y%,Z%,Radius%,Hoehe%,Anf_winkel%,End_winkel%,
Farbe%)
  Inc Fl%
  Start%(Fl%)=Pkt%+1
  Start_%=Start%(Fl%)
  '
  Inc Pkt%
  X(Pkt%)=X%
```

```
Y(Pkt%)=Y%
Z(Pkt%)=Z%+Hoehe%
'
Alpha%=Anf_winkel%
Repeat
  Inc Pkt%
  X(Pkt%)=X%+Sin(Alpha%/180*Pi)*Radius%
  Y(Pkt%)=Y%+Cos(Alpha%/180*Pi)*Radius%
  Z(Pkt%)=Z%+Hoehe%
  Alpha%=(Int(Alpha%/10)+1)*10
Until Alpha%>=End_winkel%
Alpha%=End_winkel%
Inc Pkt%
X(Pkt%)=X%+Sin(Alpha%/180*Pi)*Radius%
Y(Pkt%)=Y%+Cos(Alpha%/180*Pi)*Radius%
Z(Pkt%)=Z%+Hoehe%
'
Anz%(Fl%)=Pkt%-Start%(Fl%)+1
Anz_%=Anz%(Fl%)
Farbe%(Fl%)=Farbe%
For I%=Start_% To Start_%+Anz_%-1
  Inc Fl%
  Start%(Fl%)=Pkt%+1
  Anz%(Fl%)=4
  Farbe%(Fl%)=Farbe%+1
  '
  Inc Pkt%
  X(Pkt%)=X(I%)
  Y(Pkt%)=Y(I%)
  Z(Pkt%)=Z%
  '
  Inc Pkt%
  X(Pkt%)=X(I%)
  Y(Pkt%)=Y(I%)
  Z(Pkt%)=Z%+Hoehe%
  '
  If I%=Start_%+Anz_%-1
    Inc Pkt%
    X(Pkt%)=X(Start_%)
    Y(Pkt%)=Y(Start_%)
    Z(Pkt%)=Z%+Hoehe%
    '
    Inc Pkt%
    X(Pkt%)=X(Start_%)
    Y(Pkt%)=Y(Start_%)
    Z(Pkt%)=Z%
```

```
    Else
      Inc Pkt%
      X(Pkt%)=X(I%+1)
      Y(Pkt%)=Y(I%+1)
      Z(Pkt%)=Z%+Hoehe%
      '
      Inc Pkt%
      X(Pkt%)=X(I%+1)
      Y(Pkt%)=Y(I%+1)
      Z(Pkt%)=Z%
    Endif
    '
  Next I%
Return
```

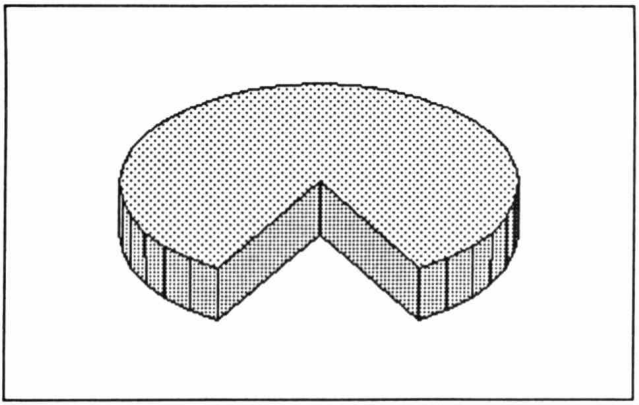


Abb. 31: Torte

5.6.3 Zeichenroutine für perspektivische Diagramme

```
Procedure Zeige_grafik(Entfernung%,Winkel%)
  ' ##### Z-Achse drehen -> Drehung
  Cos_drehung=Cos(150/180*Pi)
  Sin_drehung=Sin(150/180*Pi)
  Clr I%
  Repeat
    Inc I%
    X_=X(I%)
```

```
  X(I%)=Cos_drehung*X_-Sin_drehung*Y(I%)
  Y(I%)=Cos_drehung*Y(I%)+Sin_drehung*X_
Until I%=Pkt%
' ##### X-Achse drehen -> Winkel
Cos_winkel=Cos(Winkel%/180*Pi)
Sin_winkel=Sin(Winkel%/180*Pi)
Clr I%
Repeat
  Inc I%
  Y_=Y(I%)
  Y(I%)=Cos_winkel*Y_-Sin_winkel*Z(I%)
  Z(I%)=Cos_winkel*Z(I%)+Sin_winkel*Y_
Until I%=Pkt%
' ##### X_monitor, Y_monitor berechnen
Clr I%
Repeat
  Inc I%
  X_monitor%(I%)=(X(I%)*Entfernung%)/(Entfernung%+Y(I%))+320
  Y_monitor%(I%)=-(Z(I%)*Entfernung%)/(Entfernung%+Y(I%))+200
  Pkt_entf(I%)=Sqr(X(I%)^2+(Entfernung%+Y(I%))^2+Z(I%)^2)
Until I%=Pkt%
' ##### minimale Entfernung berechnen
Clr I%
Repeat
  Inc I%
  Entfernung(I%)=10000000
  For J%=Start%(I%) To Start%(I%)+Anz%(I%)-1
    If Pkt_entf(J%)<Entfernung(I%)
      Entfernung(I%)=Pkt_entf(J%)
    Endif
  Next J%
Until I%=Fl%
Clr I%
Repeat
  Inc I%
  Clr Summe
  For J%=Start%(I%) To Start%(I%)+Anz%(I%)-1
    Add Summe,Pkt_entf(J%)
  Next J%
  Add Entfernung(I%),Summe/Anz%(I%)
Until I%=Fl%
' ##### Sortieren nach Entfernung
For J%=Fl% Downto 1
  Min=10000000
  For I%=1 To J%
    If Entfernung(I%)<Min
```

```
        Min=Entfernung(I%)
        Index%=I%
      Endif
    Next I%
    '
    Swap Start%(J%),Start%(Index%)
    Swap Anz%(J%),Anz%(Index%)
    Swap Farbe%(J%),Farbe%(Index%)
    Swap Entfernung(J%),Entfernung(Index%)
    '
  Next J%
  ' ##### Flächen von hinten nach vorne aufbauen
  Clr I%
  Repeat
    Inc I%
    Clr J%
    Repeat
      Xx%(J%)=X_monitor%(Start%(I%)+J%)
      Yy%(J%)=Y_monitor%(Start%(I%)+J%)
      Inc J%
    Until J%=Anz%(I%)
    Deffill 1,2,Farbe%(I%)
    Polyfill Anz%(I%),Xx%(),Yy%()
  Until I%=Fl%
Return
```

Was kann ich damit nun machen? Schauen Sie sich einmal die beiden folgenden Bilder an. Diese Diagramme, in ein Zeichenprogramm übernommen, weiterbearbeitet und dann präsentiert, wirken für sich!

Das erste Beispiel zeigt das Balkendiagramm:

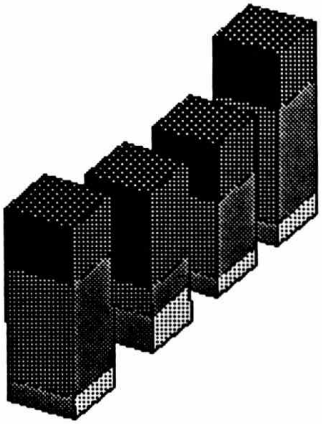


Abb. 32: **Balkengrafik**

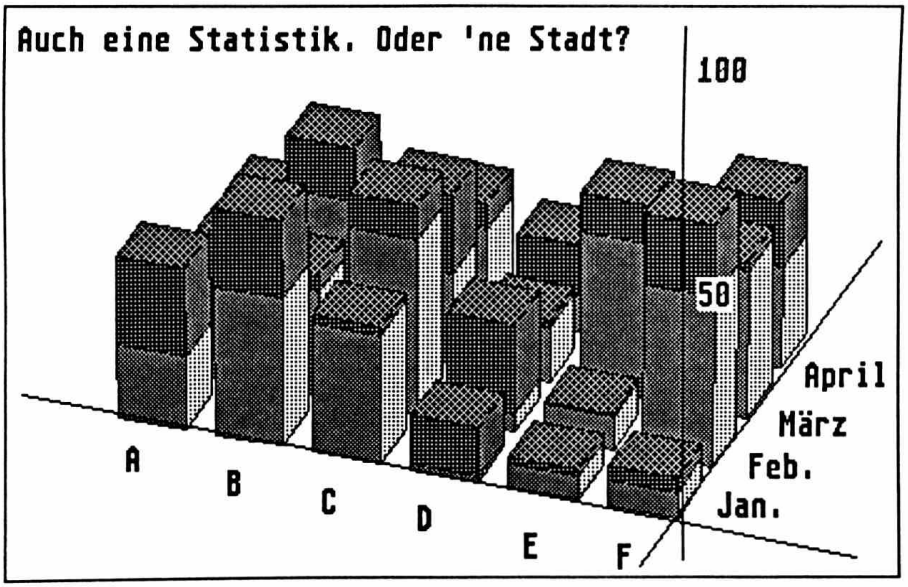


Abb. 33: **Und so könnte eine Balkengrafik nach weiterer Bearbeitung mit einem Malprogramm oder eigenen BASIC-Routinen aussehen.**

```
Dim W(20)
Data 10,40,30,20
Data 15,50,12,60
Data 30,20,20,40
For N%=1 To 9
  Read W(N%)
Next N%
'
@Def_quader(50,-5,0,35,W(7),2)
@Def_quader(50,-5,W(7),35,W(8),4)
@Def_quader(50,-5,W(7)+W(8),35,W(9),6)
@Def_quader(50,50,0,35,W(1),2)
@Def_quader(50,50,W(1),35,W(2),4)
@Def_quader(50,50,W(1)+W(2),35,W(3),6)
@Def_quader(50,110,0,35,W(4),2)
@Def_quader(50,110,W(4),35,W(5),4)
@Def_quader(50,110,W(4)+W(5),35,W(6),6)
@Def_quader(50,180,0,35,W(7),2)
@Def_quader(50,180,W(7),35,W(8),4)
@Def_quader(50,180,W(7)+W(8),35,W(9),6)
@Zeige_grafik(10000,30)
Do
  Exit If Mousek
Loop
End
```

Das zweite Beispiel baut eine Tortengrafik auf:

```
@Def_torte(0,0,0,150,50,0,50,1)
@Def_torte(30,0,0,150,50,50,120,2)
@Def_torte(0,0,0,150,50,120,220,3)
@Def_torte(0,0,0,150,50,220,250,4)
@Def_torte(0,0,0,150,50,250,300,5)
@Def_torte(0,0,0,150,50,300,360,6)
'
@Zeige_grafik(10000,30)
Do
  Exit If Mousek
Loop
End
```

Das Ergebnis nach der Aufbereitung durch ein Malprogramm sehen Sie in der folgenden Abbildung.

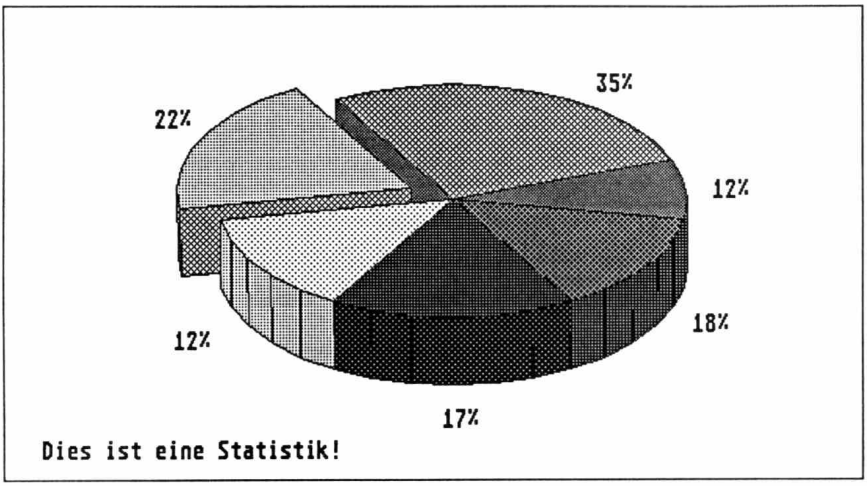


Abb. 34: Tortendiagramm

5.7. Variablen-Referenz

Den meisten von Ihnen wird bekannt sein, daß jedes '.BAS'-File über einen Vorspann verfügt, in welchem mehrere wichtige Daten vom Interpreter abgelegt werden.

Nur wenigen wird bekannt sein, wie dieser Header aufgebaut ist. Machen Sie sich keine Hoffnung, ich werde es Ihnen auch nicht verraten. Aus einem einfachen Grund. Mit genauen Kenntnissen über den Aufbau des Headers wäre es spielend möglich, mit 'PSAVE' geschützte Programme zu knacken. Da ich nicht davon ausgehe, daß schon jeder über einen GFA-BASIC-Compiler verfügt, muß diese Möglichkeit der Programmsicherung erhalten bleiben.

Was allerdings kein Geheimnis ist, ist die Organisation des Bereichs, in dem das BASIC die verwendeten Variablen-, Funktions-, Label- und Prozedurnamen abspeichert. Mit diesen Kenntnissen ist es wenigstens möglich, sich relativ leicht eine Art Referentliste über die verwendeten Namen zu erstellen.

Ab dem 10. Byte jeder BAS-Datei liegen 13 Longwords, die der Reihe nach die Offsets der einzelnen Variablentypen-Blöcke zum Byte 126 der Datei enthalten. Ab Byte 126 werden nämlich 12 Blöcke gespeichert, die die Variablen-, bzw. Label-, Prozedur- und Funktionsnamen enthalten.

In den Bytes 11 bis 14 steht also der erste Offset, der in jedem Fall 0 ist, da ja der Offset sich auf Byte 126 bezieht und der erste Block bei Byte 126 beginnt.

Dieser erste Block enthält alle Real-Variablennamen, die im Programm verwendet wurden. Daran schließen sich die Offsets für String-Variablennamen, Integervariablennamen, Boolvariablennamen, Real-Feldnamen, String-Feldnamen, Integer-Feldnamen, Bool-Feldnamen, Labelnamen, Prozedurnamen, String-Funktionsnamen und Arithmetik-Funktionsnamen an.

Das 13. und letzte Longword dieser Reihe enthält den Offset vom 126. Byte zum ersten Byte des Programmlistings.

Das war's eigentlich schon.

Mit diesen Informationen lassen sich schon fast alle Namen feststellen. Wenn da nicht noch ein Problem wäre, nämlich die Längen der einzelnen Namen, ohne die natürlich nicht viel auszurichten ist.

Ich habe bei der Block-Analyse hier die Namenslängen vernachlässigt, indem ich einfach alle Zeichen, die außerhalb des für Variablennamen zulässigen Bereich liegen ausklammere. Wo die Namenslängen eingetragen sind, soll hier aus oben genannten Grund auch nicht verraten werden.

Trotzdem können wir nun alle Namen selektieren und ein komplette Liste davon erstellen.

Das folgende Programm tut dies und schreibt alle gefundenen Namen als REM-Zeilen in eine Datei mit dem Namen 'VARS.LST'. Von dort kann diese Liste mit 'Merge' in den Arbeitsspeicher geladen und evtl. weiterverarbeitet werden.

Der Ausbau dieses Programms zu einem Programm, das eine echte Referenzliste aller verwendeten Variablen (evtl. mit Zeilennummern oder nach Prozeduren sortiert) enthält, dürfte keinen allzugroßen Aufwand mehr darstellen.

```
' Programm: BAS_HEAD.BAS
'
|''''''
```

```
  Open "O",#1,"VARS.LST"          ! LST-File öffnen
  Print #1;Vblk$                  ! Rem-Zeilen-Puffer schreiben
  Close
Endif
Edit
Data "Realvariablen          "
Data "Stringvariablen $        "
Data "Integervariablen %         "
Data "Boolvariablen !        "
Data "Realfelder ()        "
Data "Stringfelder $()       "
Data "Integerfelder %()       "
Data "Boolfelder !()       "
Data "Labels :        "
Data "Prozeduren  Procedure"
Data "Numerische Funktionen  (FN)     "
Data "Stringfunktionen $ (FN$)   "
Data
```

5.8 Der verboxte Screen

Wie oft steht man vor dem Problem, daß man verschiedene Pboxen oder Boxen auf dem Bildschirm systematisch oder symmetrisch aufzubauen hat, um z.B. Fenster zu umrahmen oder Auswahlboxen anzubieten. Meistens artet das dann in guter alter 'Trial and Error'-Tradition (Versuch und Irrtum) dahin aus, daß man in mehrmaligen Versuchen die Eck-Koordinaten dieser Boxen bestimmt. Dieses kleine Programm soll nun das lästige Hin und Her zwischen Boxdaten-Änderung und Test unnötig machen.

Da diese Boxen oft in einen schon bestehenden Screen-Aufbau eingebaut werden müssen, können Sie in diesem Fall vor Programmstart im Direktmodus den Befehl '@Xcl(0)' eingeben. Dadurch wird erreicht, daß der Bildschirm bei Programmstart nicht gelöscht wird und Sie die Boxen in einen beliebigen, vorher zu erzeugenden Screen-Aufbau einfügen können (s. 'Spezial-Adressen').

Um eine Box zu erzeugen, drücken Sie eine Maustaste, wodurch eine der Box-Ecken fixiert ist. Ziehen Sie nun die Box mit

weiterhin gedrückter Maustaste auf. Lassen Sie die Maustaste los, befindet sich die definierte Box in der Schwebe und kann an einer beliebigen Stelle plaziert werden. Nachdem Sie die Position der Box bestimmt haben, drücken Sie nochmal eine der beiden Maustasten, und die Box wird fixiert (gesetzt).

Das Setzen der Boxen erfolgt bei leeren Boxen mit der linken und bei vollen Pboxen mit der rechten Maustaste. Da diese Boxen dann als Objekte in einem Feld abgelegt werden, können Sie beliebige Boxen auch wieder löschen, wenn Sie meinen, daß ihre Position doch nicht richtig gewählt ist. Dazu fahren Sie mit dem Mauskreuz exakt(!) auf eine Ecke der zu löschenden Box, drücken die <Delete>-Taste, und die Box verschwindet. Eine andere Möglichkeit des Löschens besteht darin, daß die <Undo>-Taste gedrückt wird, womit Sie jeweils die letzte Box löschen.

Außerdem können Sie die zuletzt gesetzte Box beliebig oft kopieren, indem Sie die <Tab>-Taste und dann die jeweilige Maustaste (Box/Pbox) drücken. Haben Sie die <Tab>-Taste gedrückt, jedoch die Box noch nicht mit der Maustaste fixiert, verschwindet die kopierte Box wieder mit nochmaligem Druck auf die <Tab>-Taste.

Möchten Sie irgendeine andere Box kopieren, fahren Sie wieder mit dem Mauskreuz exakt(!) auf eine Ecke der zu kopierenden Box und drücken dann die <Insert>-Taste. Die neue Box erscheint und kann wieder mit einer der Maustasten gesetzt werden. Ist sie noch beweglich, verschwindet Sie wieder, indem Sie die <Tab>-Taste drücken.

Drücken Sie dagegen die <Ctlrhome>-Taste, werden alle Boxen gelöscht, und das Programm startet neu.

Um den Komfort noch etwas abzurunden, können Sie eine Box (solange Sie noch im 'Schwebezustand' ist) mit den vier Cursor-Pfeiltasten in der Größe beliebig verändern.

Damit Sie während des Programms nicht hilflos sind, können Sie nun noch mit einem Druck auf die <Help>-Taste ein Info-Fenster aufrufen, das Ihnen sämtliche Programm-Funktionen anzeigt.

Haben Sie den Screen erstellt, drücken Sie die <Esc>-Taste, womit das Programm beendet und die Boxdaten ggf. in einer LST-Datei als 'BOX'/'PBOX'-Befehlszeilen oder als Data-Zeilen abgelegt werden. Den Data-Zeilen ist dann eine Identifikations-Ziffer vorangestellt, welche Auskunft über den Box-Typ gibt (1=Box / 2=Pbox).

Als Parmeter übergeben Sie der Hauptprozedur die Anzahl der maximal zu zeichnenden Boxen (hier: 100), damit das Objekt-Feld eingerichtet werden kann. Außerdem wird eine Pointer-Variable erwartet, die nach Prozedurende die tatsächlich gezeichnete Anzahl enthält (Evtl. vorher @Xcl(0) im Direktmodus eingeben).

```
' Programm: SET_BOXS.BAS
'
' |''''''
```

```
Repeat                          ! Auf Taste, Mausklick oder
  '                             ! Abbruchflag warten
  A%=Asc(Right$(Inkey$))
Until Mousek Or A% Or E.flg%
Repeat
Until Len(Inkey$)=0             ! Tastaturpuffer löschen
If A%=27                        ! <Esc>-Taste = Ende?
  B%=A%
  Goto Off                      !  dann zum Ausgang
Endif
If A%=98                        ! <Help>-Taste?
  @Help                         !  Hilfe-Screen  ausgeben
  Goto N_ext
Endif
Graphmode 3
If A%=71                        ! <ClrHome>-Taste?
  Alert 2,"Alle Boxen löschen?",2," OKAY | NEIN ",Bcl%
  If Bcl%=1
    Sput Screen$                !  alte Screen restaurieren
    Run                         !  Programm neu starten
  Else
    Goto N_ext
  Endif
Endif
If A%=97                        ! <Undo>-Taste ?
  If I%>0                       !  Boxen vorhanden?
    Graphmode 1
    Sput Screen$                !  alte Screen
    Dec I%                      !  Index -1
    For J%=1 To 2               !  Boxtyp
      @Kill(J%,I%)              !   Box-Objektdaten löschen
    Next J%
    @Draw.mat                   !  alle anderen Boxen neu
    Graphmode 3
    Goto N_ext
  Else                          ! keine Boxen!
    Sput Screen$                !  alte Screen restaurieren
    Goto N_ext                  !  und weiter
  Endif
Endif
If A%=127 Or A%=82              ! <Delete> oder <Insert>-Taste?
  For J%=0 To I%-1              ! alle Boxen durchgehen
    For L%=1 To 2               !  beide Boxtypen
      If Mousex=K%(L%,J%,1) Or Mousex=K%(L%,J%,3)
```

```
          If Mousey=K%(L%,J%,2) Or Mousey=K%(L%,J%,4)
            '                    !    Mauszeiger exakt auf einer
            '                    !    Ecke einer Objektbox?
            Flag%=L%             !    Boxtyp weitergeben
          Endif
        Endif
      Next L%
      If Flag%>0                 !  Box gefunden?
        If A%=127                !    <Delete>-Taste ?
          Graphmode 1
          @Kill(Flag%,J%)        !      Objekt-Eintrag löschen
          Sput Screen$           !      alte Screen
          @Draw.mat              !      alle anderen Boxen neu
          Graphmode 3
        Else                     !    <Insert>-Taste!
          X1%=K%(Flag%,J%,1)     !      -- Boxdaten
          Y1%=K%(Flag%,J%,2)     !         der gewählten
          X2%=K%(Flag%,J%,3)     !         Box werden
          Y2%=K%(Flag%,J%,4)     !      -- aktuelle Daten
        Endif
        Flag%=0
      Endif
    Next J%
    If A%=127
      Goto N_ext
    Endif
  Endif
  If A%=9 Or A%=82               ! <Tab>- oder <Insert>-Taste?
    Goto Copy                    !  dann weiter zur Boxpositionierung
  Endif
  @Drawbox                       ! neue Box zeichen
  Copy:
  @Fixbox                        ! Box positionieren
  If (K%=1 Or K%=2) And B%<>9
    @Initarray                   ! Box ins Objekt-Array eintragen
  Endif
  Off:
Until B%=27 Or E.flg%            ! <Esc> = Abbruch
Al$="Programm-Ende !|Box-Datas abspeichern ?"
Alert 2,Al$,1," OKAY | NEIN | CONT ",B%
If B%=1
  *Fakt%=I%
  @Saving
Endif
```

```
  If B%=3
    Goto Start
  Endif
  @Xcl(1)                          ! Programmstart-Cls wieder an
Return
Procedure Drawbox                  ! neue Box zeichnen
  Mouse X1%,Y1%,K%
  Repeat
    Mouse X2%,Y2%,K%
    Box X1%,Y1%,X2%,Y2%
    Repeat
      Mouse X4%,Y4%,K%
    Until X2%<>X4% Or Y2%<>Y4% Or K%=0
    Box X1%,Y1%,X2%,Y2%
  Until K%=0 Or E.flg%
Return
Procedure Fixbox                   ! Box positionieren
  Repeat
    Mouse X3%,Y3%,K%
    Box X3%-(X2%-X1%),Y3%-(Y2%-Y1%),X3%,Y3%
    Repeat
      Mouse X4%,Y4%,K%
      B%=Asc(Right$(Inkey$))
    Until X3%<>X4% Or Y3%<>Y4% Or K% Or B% Or E.flg%
    Box X3%-(X2%-X1%),Y3%-(Y2%-Y1%),X3%,Y3%
    If B%=75                       ! <Pfeil-links>-Taste?
      Dec X1%                      ! X-Koordinate -1
    Endif
    If B%=72                       ! <Pfeil-hoch>-Taste?
      Dec Y1%                      ! Y-Koordinate -1
    Endif
    If B%=77                       ! <Pfeil-rechts>-Taste?
      Inc X1%                      ! X-Koordinate +1
    Endif
    If B%=80                       ! <Pfeil-runter>-Taste?
      Inc Y1%                      ! Y-Koordinate +1
    Endif
  Until K% Or B%=9 Or B%=27 Or E.flg%
Return
Procedure Initarray                ! Objektdaten eintragen
  Graphmode 1
  Xlo$=Str$(X3%-(X2%-X1%))         ! X-Ecke links oben | für Befehls-
  Ylo$=Str$(Y3%-(Y2%-Y1%))         ! Y-Ecke links oben |    zeile
  If K%=2                          ! rechte Maustaste?
    Pbox X3%-(X2%-X1%),Y3%-(Y2%-Y1%),X3%,Y3% ! dann Pbox zeichnen
```

```
    B$(I%)="Pb "+Xlo$+","+Ylo$+","+Str$(X3%)+","+Str$(Y3%) ! Befehls-
    '                                                      !  Zeile
  Else                              ! linke Maustaste!
    Box X3%-(X2%-X1%),Y3%-(Y2%-Y1%),X3%,Y3%  ! dann Box zeichnen
    B$(I%)="B "+Xlo$+","+Ylo$+","+Str$(X3%)+","+Str$(Y3%)  ! Befehls-
    '                                                      !  Zeile
  Endif
  K%(K%,I%,1)=X3%-(X2%-X1%)         ! Daten ins Feld eintragen
  K%(K%,I%,2)=Y3%-(Y2%-Y1%)         !
  K%(K%,I%,3)=X3%                   !
  K%(K%,I%,4)=Y3%                   !
  Inc I%                            ! Box-Zähler +1
  Pause 15
Return
Procedure Kill(J%,I%)               ! Box im Feld löschen
  K%(J%,I%,1)=0                     !  Daten auf Null
  K%(J%,I%,2)=0                     !
  K%(J%,I%,3)=0                     !
  K%(J%,I%,4)=0                     !
Return
Procedure Draw.mat                  ! alle Boxen zeichnen
  For Z%=0 To I%-1
    Box K%(1,Z%,1),K%(1,Z%,2),K%(1,Z%,3),K%(1,Z%,4)
    Pbox K%(2,Z%,1),K%(2,Z%,2),K%(2,Z%,3),K%(2,Z%,4)
  Next Z%
Return
Procedure Saving                    ! Zeilen abspeichern
  Al$="Als Box/Pbox-Zeilen|oder als Data-Zeilen?"
  Alert 2,Al$,0,"(P)Box|Datas",Bk%
  Fileselect "\*.LST",".LST",Boxlst$
  If Boxlst$>"" And Boxlst$<>"\.LST" And Boxlst$<>"\"
    Open "O",#99,Boxlst$
    For J%=0 To I%
      If Bk%=2
        For M%=1 To 2
          If K%(M%,J%,1)>0 And K%(M%,J%,2)>0
            If K%(M%,J%,3)>0 And K%(M%,J%,4)>0
              Print #99;"D ";M%;",";K%(M%,J%,1);",";K%(M%,J%,2);
              Print #99;",";K%(M%,J%,3);",";K%(M%,J%,4)
            Endif
          Endif
        Next M%
      Else
        Print #99,B$(J%)
      Endif
```

```
    Next J%
    Close #99
  Endif
Return
Procedure Help
  Graphmode 1
  Get 100,75,540,325,H.scr$
  Pbox 100,75,540,325
  Pbox 102,77,538,323
  Print At(20,7);"S   E   T   B   O   X   -   I   N   F   O"
  Print At(17,9);"Maustaste  links  = leere Box"
  Print At(17,10);"Maustaste rechts  = weiß gefüllte Pbox"
  Print At(17,11);"<UNDO>    - Taste  = letzte Box löschen"
  Print At(17,12);"<DELETE> - Taste  = Box unter Mauszeiger löschen"
  Print At(17,13);"<CLRHOME>- Taste  = alle Boxen löschen"
  Print At(17,14);"<TAB>     - Taste  = letzte Box kopieren"
  Print At(17,15);"<INSERT> - Taste  = Box unter Mauszeiger kopieren"
  Print At(17,16);"<CURSOR> - Tasten = Schwebebox-Größe variieren"
  Print At(17,17);"<HELP>    - Taste  = dieses INFO"
  Print At(17,18);"<ESC>     - Taste  = Programmende"
  Print At(35,20);"T A S T E"
  Repeat
  Until Len(Inkey$)
  Put 100,75,H.scr$
Return
Procedure Xcl(Flg%) ! flg%=0=cls off / flg%<>0=cls on
  Local A$
  A$=Space$(3000)
  Bmove Basepage,Varptr(A$),3000
  If Flg%=0
    Poke Basepage+Instr(A$,Chr$(27)+"E"),Asc("H")
  Else
    Poke Basepage+Instr(A$,Chr$(27)+"H"),Asc("E")
  Endif
Return
Procedure Ende
  E.flg%=1
  Cont
Return
```

5.9 "Kalkuliere....."

Da das GFA-BASIC, wie Sie auch durch dieses Buch erfahren können, ein nahezu selbständiges Entwicklungssystem ist, dürfen natürlich auch die kleinen Bonbons, die einer solchen Sprache zustehen, nicht fehlen.

Dieser kleine 'Taschenrechner' ist in seinem Format und seinen Funktionen so gestaltet, daß er bequem in jeder Tasche Platz hat und trotzdem für die üblichsten Operationen verwendbar ist.

Da er sogar über eine Prozent- und Quadrierungsautomatik verfügt, wird er auch den etwas höheren Ansprüchen gerecht. Die Potenz- und Wurzelarithmetik nicht zu vergessen.

Sein Aufruf und seine Bedienung sind denkbar einfach.

Auch in diesem Listing finden Sie einige nützliche Anregungen zur Weiter- Entwicklung in eigenen Programmen.

```
|''''''
```

1-23	23 Punkt-Füllmuster
24-36	13 Linien-Füllmuster
37-47	Diagonalstreifen
48-71	ATARI-, bzw. selbstdefiniertes Füllmuster
71-...	weiß

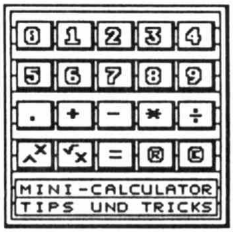


Abb. 35: GFA-'Taschen'-Rechner

```
Edit
Procedure Calc(L%,O%,F%,C%)
  Restore D.atas                          ! Zeiger auf Tastendatas
  Local Back$,Back2$,Op1$,Op2$,Funkt._done%,Ergebnis,R$
  Local Xo%,Yo%,Funktion%,I%,J%,K%,X%,Y%,Xp%,Yp%,Tast._index%
  If L%=0 Or O%=0                         ! X/Y-Parameter 0?
    L%=Mousex                             !   X-Koordinate
    O%=Mousey                             !   Y-Koordinate
  Endif
  If (L%-55 Or O%-55)<0 Or L%+55>639 Or O%+55>399
    '                 ! Liegt der Rechner außerhalb des Bildschirms?
    L%=320                                !  dann Rechner in Bild-
    O%=200                                !  mitte positionieren
  Endif
  Dpoke Gintin,L%                 !-------.
  Dpoke Gintin+2,O%                       !
  Dpoke Gintin+4,2                        !
  Dpoke Gintin+6,2                        !------- AES-Growbox
  Dpoke Gintin+8,L%-55                    !
  Dpoke Gintin+10,O%-55                   !
  Dpoke Gintin+12,110                     !
  Dpoke Gintin+14,110                     !
  Gemsys 73                       !-------'
  Graphmode 1
  Deftext 1,0,0,13
```

```
Get L%-55,O%-55,L%+56,O%+56,Back$          ! Hintergrund sichern
Deffill ,0,0
Pbox L%-55,O%-55,L%+55,O%+55             !-.
Box L%-55,O%-55,L%+56,O%+56                !
Deffill ,F% Div 24+2,F% Mod 24             !
Pbox L%-52,O%-52,L%+52,O%+52               !--- Rechner zeichnen
Deffill ,0,0                               !
Pbox L%-50,O%+32,L%+50,O%+50             !-'
Deftext 1,C%,0,6
For I%=1 To 4                                        !--.
  For J%=1 To 5                                        !
    Pbox L%-69+J%*20,O%-67+I%*20,L%-51+J%*20,O%-53+I%*20!
    Box L%-69+J%*20,O%-67+I%*20,L%-52+J%*20,O%-54+I%*20 !
    If I%<3                                            !
      Graphmode 2                                      !-- Tasten
      Text L%-65+J%*20,O%-57+I%*20,Chr$(47+(I%-1)*5+J%) !    zeichnen
      Graphmode 1                                      !
    Endif                                              !
  Next J%                                              !
Next I%                                              !-'
Graphmode 2
Deftext ,0,,6
For K%=1 To 12                                       !--.
  Read X%,Y,Z%                                          !
  Z$=CHR$(Z%)                                           !- Funktions-
  Text L%-71+X%,O%+2+Y*10,Z$                            !  Tasten
Next K%                                              !--'
Graphmode 2
Line L%-50,O%+41,L%+50,O%+41                         !--.
Deftext ,0,,4                                           !
Text L%-47,O%+39,96,"MINI-CALCULATOR"                   !- Ausgabe-
Text L%-47,O%+48,96,"TIPS UND TRICKS"                !--'  Fenster
Pause 40
For K%=1 To 8                                        !---.
  Get L%-47,O%+34,L%+47,O%+40,R1$                       !
  Get L%-47,O%+43,L%+47,O%+48,R2$                       !
  Pause 3                                               !- Scroll-
  Put L%-47,O%+33,R1$,3                                 !  Trick
  Put L%-47,O%+44,R2$,3                                 !
Next K%                                               !--'
Text L%+43,O%+47,"CHR$(9)"                            ! Help-Button

Box L%-68+(J%-1)*20,O%-66+(I%-1)*20,L%-53+(J%-1)*20,O%-55+(I%-1)*20
Box L%-68+(J%-2)*20,O%-66+(I%-1)*20,L%-53+(J%-2)*20,O%-55+(I%-1)*20
Box L%-67+(J%-3)*20,O%-65+(I%-1)*20,L%-54+(J%-3)*20,O%-56+(I%-1)*20
Graphmode 1
```

```
@Sweep                                  ! Ausgabe-Fenster löschen
Defnum 10                               ! Werte-Ausgabe begrenzen
Repeat                                  ! Hauptschleife
  Mouse X%,Y%,K%                        !  Mausstatus
  Xp%=Int((X%-(L%-50))/20)+1            !  Tasten-X-Index berechnen
  Yp%=Int((Y%-(O%-50))/20)+1            !  Tasten-Y-Index berechnen
  If K%>0                               !  Maustaste gedrückt?
    If Xp%=5 And Yp%=5 And Y%>O%+42 And X%>L%+41!Help-Button gewählt?
      @Help                             !   Helptext zeigen
    Endif
    If Xp%>0 And Xp%<6 And Yp%>0 And Yp%<5!  Taste angeklickt?
      Deffill ,1,1
      Graphmode 3
      Xo%=(Xp%-1)*20                    !   X-Blinkbox-Offset
      Yo%=(Yp%-1)*20                    !   Y-Blinkbox-Offset
      Pbox L%-48+Xo%,O%-46+Yo%,L%-33+Xo%,O%-35+Yo% !--.
      Pause 6                                          !--Blinkbox
      Pbox L%-48+Xo%,O%-46+Yo%,L%-33+Xo%,O%-35+Yo% !--'
      Graphmode 1
      Deffill ,0,0
      Tast._index%=(Yp%-1)*5+Xp%        !   Absoluter Tastenindex
      If Tast._index%>0 And Tast._index%<11! Zifferntaste gewählt?
        @Sweep                          !   Ausgabefenster löschen
        If Funkt._done%=0               !   Noch keine Funktion?
          If Len(Op1$)<15               !    1.Wert < 15 Zeichen?
            Op1$=Op1$+Str$(Tast._index%-1)!    Wertstring bilden
            Text L%-47+96-Len(Op1$)*6,O%+39,Op1$ ! und ausgeben
          Else                          !    String > 14 Zeichen!
            @Long(Op1$)                 !     Fehlermeldung!
          Endif
        Else                            !  Funktion schon bestimmt!
          If Len(Op2$)<15               !   2. Wert < 15 Zeichen?
            Op2$=Op2$+Str$(Tast._index%-1)!   Wertstring bilden
            If Funkt._mem%=1 And Funktion%=3 !letzte Funktion war '+'
              '                         !   und neue F. ist '*' ?
              Text L%-47+96-(Len(Op2$)+1)*6,O%+39,Op2$+"%" ! Wert +
              '                         !     Prozentzeichen ausgeben
            Else                        !   sonst
              Text L%-47+96-Len(Op2$)*6,O%+39,Op2$! 2.Wert ausgeben
            Endif
          Else                          ! 2.Wert >14 Zeichen!
            @Long(Op2$)                 !   Fehlermeldung
          Endif
        Endif
      Endif
```

```
If Tast._index%=13                  ! Minus-Zeichen gewählt?
  If Funkt._done%=0 And Op1$=""     !  Funktion und 1.Wert noch
    '                               !  nicht bestimmt?
    @Sweep                          !   Ausgabefenster löschen
    Op1$="-"                        !   erstem Wert ein Minus-
    '                               !   Zeichen voranstellen
    Text L%-47+96-Len(Op1$)*6,0%+39,Op1$! und ausgeben
    Tast._index%=0                  !   sonst nichts tun
  Endif
  '
  If Funkt._done%=1 And Op2$=""     !  Funktion bestimmt, aber
    '                               !  noch kein 2. Wert?
    @Sweep                          !   Ausgabefenster löschen
    Op2$="-"                        !   zweitem Wert ein Minus-
    '                               !   Zeichen voranstellen
    Text L%-47+96-Len(Op2$)*6,0%+39,Op2$! und ausgeben
    Tast._index%=0                  !   sonst nichts tun
  Endif
Endif
If Tast._index%>11 And Tast._index%<18! Funkt.-taste angeklickt?
  Funkt._mem%=Funktion%             !   letzte Funktion merken
  Funktion%=Tast._index%-11         !   neue Funktion bestimmen
  If Tast._index%=17                !   Wurzel(X) gewählt?
    Tast._index%=18                 !    dann zugleich 'Gleich'-
    '                               !    Taste aktivieren
  Endif
  Funkt._done%=1                    !   Funktion ist aktiv!
Endif
If Tast._index%=11                  !  Dezimal-Punkt angeklickt?
  If Funkt._done%=0 And Instr(Op1$,".")=0!Noch keine Funktion
    '                               !    aktiv und 1.Wert enthält
    '                               !    noch keinen Dez.-Punkt?
    @Sweep                          !     Ausgabefenster löschen
    Op1$=Op1$+"."                   !     Punkt an 1.Wert anfügen
    Text L%-47+96-Len(Op1$)*6,0%+39,Op1$! und 1.Wert ausgeben
  Endif
  If Funkt._done%=1 And Instr(Op2$,".")=0!Funktion schon aktiv,
    '                               !    aber 2.Wert enthält
    '                               !    noch keinen Dez.-Punkt?
    @Sweep                          !     Ausgabefenster löschen
    Op2$=Op2$+"."                   !     Punkt an 2.Wert anfügen
    If Funkt._mem%=1 And Funktion%=3 !   letzte Funktion war '+'
      '                             !     und neue F. ist '*' ?
      Text L%-47+96-(Len(Op2$)+1)*6,0%+39,Op2$+"%" ! Wert +
      '                             !     Prozentzeichen ausgeben
```

```
      Else                                  !   sonst
        Text L%-47+96-Len(Op2$)*6,0%+39,Op2$ !2.Wert ausgeben
      Endif
    Endif
  Endif
  If Tast._index%=18                        ! 'Gleich'-Taste angeklickt?
    @Sweep                                  !  Ausgabefenster löschen
    If Funkt._done%=1 And Op2$=""           !  Funktion bereits gewählt,
      '                                     !   aber 2.Wertstring leer?
      Op2$=Op1$                             !   2.Wert = 1.Wert
```

So unscheinbar, wie diese kleine Raffinesse wirkt, so effektiv ist sie auch. Hierdurch wird es möglich, den Ergebniswert je nach gewählter Funktion zu sich selbst zu addieren, mit sich zu multiplizieren oder zu quadrieren. Das wird erreicht, indem nach Ausgabe des Ergebnisses eine Funktion (+, * oder ^) gewählt wird und statt Eingabe des zweiten Wertes einfach wieder die 'Gleich'-Taste angeklickt wird.

```
    Endif
    If Funkt._done%=1 And Op2$>""           ! Funktion gewählt und auch
      '                                     ! 2.Wert bereits eingegeben?
      If Funkt._mem%=1 And Funktion%=3 !letzte Funktion war '+'
        '                                   !   und neue F. ist '*' ?
        Ergebnis=Val(Op1$)+Val(Op1$)*Val(Op2$)/100 !zum 1.Wert
        '                                   !   (Wert 2) Prozent des
        '                                   !   1. Wertes addieren
        Funktion%=0                         !   sonst nichts tun
```

Auch diese Bedingungsabfrage hat es in sich. Werden direkt nacheinander die 'Plus'-, dann die 'Mal'-Taste geklickt, und wird anschließend ein Wert eingegeben, wird das als Prozent-Funktion interpretiert. D.h., daß bei der nächsten 'Gleich'-Wahl der erste Wert, bzw. das Ergebnis mit dem durch 100 geteilten 2. Wert multipliziert wird und dieser Wert zum ersten Wert bzw. dem Ergebniswert addiert wird.

Das Verfahren mit Minuswerten ist vergleichbar. Nur daß hier nach 'Plus'- und 'Mal'-Wahl die Minustaste angeklickt wird, um so einen Minusprozentwert einzugeben. Nach abschließender

'Gleich'-Tastenwahl wird dann der 1. Wert bzw. das Ergebnis mit dem durch 100 geteilten 2. Wert multipliziert und dieser Wert dann vom ersten bzw. dem Ergebniswert abgezogen wird.

```
  Endif
Endif
If Funkt._done%=0                !Noch keine Funktion bestimmt?
  Ergebnis=Val(Op1$)             ! Ergebnis = 1. Wert
Endif
On Error Gosub Error             ! evtl. Fehler abfangen
If Funktion%=1                   ! 'Plus'-Funktion gewählt?
  Ergebnis=Val(Op1$)+Val(Op2$)   !  Ergebnis = 1. + 2. Wert
Endif
If Funktion%=2                   ! 'Minus'-Funktion gewählt?
  Ergebnis=Val(Op1$)-Val(Op2$)   !  Ergebnis = 1. - 2. Wert
Endif
If Funktion%=3                   ! 'Mal'-Funktion gewählt?
  Ergebnis=Val(Op1$)*Val(Op2$)   !  Ergebnis = 1. * 2. Wert
Endif
If Funktion%=4                   ! 'Durch'-Funktion gewählt?
  Ergebnis=Val(Op1$)/Val(Op2$)   !  Ergebnis = 1. / 2. Wert
Endif
If Funktion%=5                   ! 'Potenz'-Funktion gewählt?
  Ergebnis=Val(Op1$)^Val(Op2$)   !  Ergebnis = 1. ^ 2. Wert
Endif
If Funktion%=6                   ! 'Wurzel'-Funktion gewählt?
  Ergebnis=Sqr(Val(Op1$))        !  Ergebnis=Wurzel aus 1.Wert
  Deffill ,1,1                   !---.
  Graphmode 3                        !
  Pbox L%-8,0%+14,L%+7,0%+25         !
  Pause 2                            !--'Gleich'-Taste
  Pbox L%-8,0%+14,L%+7,0%+25         !   betätigt (optisch)
  Graphmode 1                        !
  Deffill ,0,0                       !
Endif                            !---'
Op1$=Str$(Ergebnis)              !Ergebnis zur Weiterverarbei-
'                                !tung in 1. Wert übernehmen
Text L%-47,0%+39,Op1$            !Ergebnis ausgeben
If Ergebnis<2^31-1 And Ergebnis>2^31*(-1) ! Ergebnis im
  '                              !Integer-Bereich?
  Text L%-47,0%+48,"HEX$ "+Hex$(Ergebnis)! HEXA-Wert ausgeben
Else                             !Wert außerhalb Integer!
  Text L%-47,0%+48,87,"HEXA NUR INTEGER" !Meldung
Endif
Funkt._mem%=Funktion%            !alte Funktion merken
```

```
        Clr Funkt._done%,Ergebnis,Funktion%,Op2$! alles andere löschen
      Endif
      E.label:                          !Error-Rückkehr-Label
      On Error                          !Error-Handling normal
      If Tast._index%=19                ! (R) angeklickt?
        @Sweep                          !  Ausgabefenster löschen
        If Funkt._done%=0               !  keine Funktion gewählt?
          If Len(Op1$)>0                !   1.Wertstring > 0 ?
            Op1$=Left$(Op1$,(Len(Op1$)-1))!  letzte Ziffer abschneiden
            Text L%-47+96-Len(Op1$)*6,0%+39,Op1$!neuen 1.Wert ausgeben
          Endif
        Else                            !  Funktion schon aktiv!
          If Len(Op2$)>0                !   2.Wertstring > 0 ?
            Op2$=Left$(Op2$,(Len(Op2$)-1))!  letzte Ziffer abschneiden
            Text L%-47+96-Len(Op2$)*6,0%+39,Op2$!neuen 2.Wert ausgeben
          Endif
        Endif
      Endif
      If Tast._index%=20                 ! (C) = Clear angeklickt?
        Clr Funkt._done%,Op2$,Ergebnis,Op1$,Funktion%! alles löschen
        @Sweep                           !   Ausgabefenster löschen
      Endif
    Endif
  Endif
Until Inkey$>"" Or (K% And (Xp%>6 Or Xp%<0 Or Yp%>6 Or Yp%<0))
'                                  ! Abbruch, wenn Tastatur betätigt
'                                  ! oder Maustaste außerhalb des
'                                  ! Rechners gedrückt wurde
Put L%-55,0%-55,Back$              ! Hintergrund restaurieren
Gemsys 74                          ! AES-Shrinkbox mit alten
'                                  ! Growshrink-Parametern aufrufen
If K%=2                            ! Ausstieg mit rechter Maustaste?
  @Calc(Mousex,Mousey,Random(35)+1,Random(21))! rekursiver Aufruf
  '                                ! des Rechners an neuer Position
```

Obwohl diese Rekursion nichts Weltbewegendes an sich hat, wird daran ein Prinzip klar. Bei großen Prozeduren kann es sein, daß Variablen nicht als lokal definiert wurden. Bei komplizierteren Rekursionen kann es ohne weiteres Absicht sein, Variableninhalte von einem Aufruf zum nächsten zu erhalten. Wer sich solche Rekursionen zutraut, wird auch wissen, wie er in diesem Fall zu handeln hat. In den meisten Fällen wird eine rekursive Routine in sich geschlossen sein. Um in diesen Fällen mit der Kontrolle der Variableninhalte nicht durcheinander zu

geraten, sollten die rekursiven Aufrufe jeweils am Ende der Routine plaziert werden, da nach Abschluß des letzten Aufrufs nur noch die 'Returns' am Ende der Prozedur passiert werden müssen.

```
Hauptprogramm   ==> zur Procedure XYZ
                         Inhalt
                  ==> zur Procedure XYZ
                           Inhalt
                    ==> zur Procedure XYZ
                             Inhalt
                    <==     Return (zurück zu XYZ)
                  <==     Return (zurück zu XYZ)
Hauptprogramm   <==     Return (zurück zum Hauptprogramm)
```

```
  Endif
D.atas:
'+,-,*,v,.,^,x,=,ä,x,>,=
Data 46,0,43
Data 66,0,45
Data 86,0,42
Data 107,0.1,246
Data 26,0,46
Data 25,2.4,94
Data 30,1.7,120
Data 67,2,61
Data 45,1.7,251
Data 50,2.1,120
Data 87,2,190
Data 107,2,189
Return
Procedure Long(S.tr$)
```

Wurden mehr als 14 Zeichen eingegeben, wird hier lediglich eine Meldung dazu produziert.

```
  Text L%-47,O%+39,"ZEILE ZU LANG"
  Pause 10
  Pbox L%-49,O%+33,L%+50,O%+41
  Text L%-47+96-Len(S.tr$)*6,O%+39,S.tr$
Return
Procedure Sweep
```

An mehreren Stellen in der Prozedur wird das Löschen des Ausgabefensters notwendig. Das passiert hier.

```
  Pbox L%-49,O%+33,L%+50,O%+41
  Pbox L%-49,O%+42,L%+41,O%+50
Return
Procedure Help
```

Ebenfalls eine banale Prozedur, die jedoch (ebenso wie die Hauptroutine) ein Prinzip verdeutlicht. Die Verwendung von Data-Zeigern auf Datas innerhalb der Prozedur, wodurch die Selbständigkeit der Prozedur unterstrichen wird. Innerhalb des Hauptprogramms ist dann jedoch darauf zu achten, daß ebenfalls mit Data-Zeigern gearbeitet wird, da sonst die Folge der zu lesenden Datas durcheinander geraten kann.

In dieser Prozedur wird der kleine Help-Screen produziert, sobald der kleine Button rechts im Hexa-Ausgabefenster angeklickt wurde. Der Helptext wurde extrem kurz gehalten. Sollte Ihnen das nicht ausreichen, läßt sich leicht eine mehrseitige, ausführlichere Anzeige verwirklichen, indem Sie die Anzeigeschleife mit weiteren Texten mehrfach durchlaufen.

```
  Restore Helptxt
  Local H.tx$
  Get L%-52,O%-52,L%+52,O%+52,Back2$        ! Rechner-Image sichern
  Pbox L%-51,O%-51,L%+52,O%+52              ! Box löschen
  Graphmode 2
  For I%=1 To 16                            ! 16 Help-Zeilen
    Read H.tx$                              !   lesen
    Text L%-49,O%-50+I%*6,100,H.tx$         !   und ausgeben
  Next I%
  Pause 10
  Graphmode 1
  Repeat
  Until Mousek Or Len(Inkey$)               ! Auf Tastendruck warten
  Put L%-52,O%-52,Back2$                    ! Rechner-Image restaurieren
  Pause 10
  Helptxt:
  Data H  I  L  F  E :,re.Maust.ausserh.,=neue Position
  Data Ergebnis-Anzeige,u.dann + > * X,=X-Proz.-Addition
  Data Ergebnis-Anzeige,u.dann + > * > -X,=X-Proz.-Subtrakt.
```

```
  Data Ergebnis-Anzeige,u.dann + * ^ > =,= Erg. + * ^ Erg.
  Data (R)= CLEAR letzte Z.,(C)= CLEAR A l l e s
  Data Eingabe-Folgen,sonst wie üblich!
Return
Procedure Error
```

Bei auftretenden Systemfehlern während der Rechner-Bedienung (Überlauf, Nulldivision etc.) wird in dieser Prozedur auf den jeweiligen Error reagiert. Nichts Besonderes also. Die meisten von Ihnen werden das wahrscheinlich so handhaben, daß hier mehrere 'If'-Abfragen die Error-Nummer feststellen und dann in dem jeweiligen 'If'-Block darauf reagiert wird.

Um diese 'If'-Abfragen einzusparen, kann man folgendermaßen vorgehen: Man legt die Error-Texte in Data-Zeilen ab, liest die Datas mit einer auf die aktuelle Error-Nummer begrenzten Schleife ein und kann nun den zuletzt gelesenen Data-Text mit einer allgemein gehaltenen Ausgaberoutine auf den Bildschirm bringen.

Wenn außer dem Text noch weitere Reaktionen folgen sollen, kann man auch mit 'On Err Gosub E1,E2,E3,E4....' die behandelnde Prozedure bestimmen und diese dadurch gleichzeitig für andere Zugriffe offenhalten, ohne einen Slalom durch das vorbereitende Error-Handling laufen zu müssen.

```
  Restore E.data                          !Zeiger auf Error-Datas
  For I%=0 To Err                         ! Lese-Schleife
    Read E.rrtxt$                         !  Datas lesen
  Next I%
  Text L%-47,0%+39,E.rrtxt$               ! letztes Data ausgeben
  Pause 50                                ! kurze Pause zum Lesen
  Pbox L%-49,0%+33,L%+50,0%+41            ! Fenster wieder löschen
  Tast._index%=20                         ! Tastenindex = CLEAR
  Resume E.label                          ! bei E.label weitermachen
  E.data:
  Data NULL-DIVISION,REAL-ÜBERLAUF,,,,MINUS-RADIKAND
  Data BASIS ZU KLEIN,UNBEKANNTER FEHLER
Return
```

5.10 Extension- und Backup-Service

Eine fast unscheinbare Routine, die es aber in sich hat. Wie oft stellt sich die Aufgabe, die Richtigkeit einer Dateinamen-Eingabe zu überprüfen oder ein Backup-File anzulegen. Während der eigentliche Dateiname in den allermeisten Fällen für den Anwender frei bestimmbar ist, hat man als Programmierer doch oft ein Interesse daran, daß die Extension richtig gesetzt ist. Es nützt meist wenig, die Extension mit dem zweiten Parameterstring der Fileselect-Box vorzugeben, wenn bei der Eingabe die Extension gelöscht oder verändert wird. Um sicherzustellen, daß garantiert die richtige Extension verwendet wird, kann man nun diese Routine aufrufen.

Der Aufruf erfolgt so:

```
Fileselect "\*.ABC",".ABC",A$ ! Eingabe des Dateinamens
@Extend(A$,"ABC",*N$)         ! Extension überprüfen
Print N$
```

Wurde in der Fileselect-Box ohne Änderung einfach die OK- oder ABBRUCH-Box angeklickt, erhält man in N$ nach 'Extend'-Aufruf den Ausdruck '000' zurück. Das gleiche geschieht, wenn vom Anwender entweder die Auswahl-Zeile ersatzlos gelöscht wurde (<Esc>) oder nur die Extension geändert, aber kein Name dazu eingegeben wurde. In allen anderen Fällen wird die Extension des eingegebenen Namens mit der Vorgabe im zweiten Parameterstring des 'Extend'-Aufrufes verglichen und bei Nicht-Übereinstimmung durch die Vorgabe ersetzt. Anschließend erhalten Sie dann den gesamten Dateinamen mit evtl. geänderter Extension in 'N$' zurück. Statt 'N$' kann natürlich bei der Pointer-Übergabe jeder beliebige Stringvariablenname verwendet werden.

Bei Backup-Files kommt noch eine zweite Mini-Routine zum Einsatz. Diese Routine 'BACKUP' erwartet zwei Parameter-Strings. Der erste gibt den Namen an, unter welchem die Datei abgelegt werden soll. Der zweite bestimmt die Extension, die Sie der Datei geben wollen, die evtl. unter demselben Namen wie die neue Datei schon auf Diskette existiert und nun als Backup gesichert werden soll.

Die 'BACKUP'-Routine erledigt nun das Finden und Umbenennen mit Hilfe der 'EXTEND'-Routine, so daß Sie anschließend Ihre neue Datei anlegen können. Als Beispiel lege ich hier das obere Viertel des Bildschirms auf Diskette ab. Der Backup-Effekt wird erst deutlich, wenn Sie das Programm zweimal starten und sich die Dateien mit 'Files' im Direktmodus anschauen. Sie haben nun zwei Dateien mit gleichem Namen, aber unterschiedlicher Extension.

```
Fileselect "\*.SCR",".SCR",A$ ! Eingabe des Dateinamens
@Backup(A$,"BAK")             ! evtl. Backup anlegen
@Extend(A$,"SCR",*N$)         ! Extension überprüfen
If N$<>"000"                  ! brauchbarer Pfad + Name?
  Bsave N$,Xbios(2),8000      !  Datei anlegen
Endif

' Programm: EXTENDER.BAS
'
'
' |''''''
```

```
      If Left$(Dn$,2)<>"\."          !   Name größer als nur Extension?
        *Ps%=Left$(Pr$,Len(Pr$)-Nl%)+Left$(Dn$,Instr(Dn$,"."))+Ex$
        '                            !    Pfad + Name + Extension zurück
      Else                           !   Name besteht nur aus Extension!
        *Ps%="000"                   !    unbrauchbaren Namen zurückgeben
      Endif
    Else                             !  richtige Extension!
      *Ps%=Left$(Pr$,Len(Pr$)-Nl%)+Dn$ ! Pfad + Name zurück
    Endif
  Else                               ! ungültiger Dateiname!
    *Ps%="000"                       !  unbrauchbaren Namen zurückgeben
  Endif
Return
Procedure Backup(Pr$,Ex$)
  '
  ' Pr$ = Dateipfad u. -name
  ' Ex$ = Backup-Datei-Extension
  '
  Local Xn$
  If Exist(Pr$)                      ! Datei auf Diskette vorhanden?
    @Extend(Pr$,Ex$,*Xn$)            !  Backup-Extension einbauen
    If Xn$<>"000"                    !  brauchbarer Dateiname?
      If Exist(Xn$)                  !   Schon Backup-File vorhanden?
        Kill Xn$                     !    dann löschen
      Endif
      Name Pr$ As Xn$                !   alte Datei auf Backup umbenennen
    Endif
  Endif
Return
```

5.11 Hochleistungskompressor

Es kann manchmal sehr nützlich sein, mehr Platz auf einer Diskette schaffen zu können, ohne dabei auf die wichtigen Daten verzichten zu müssen. In den meisten Fällen ist dies nicht möglich, da die Struktur der Dateien und die Art und Weise des Zugriffs aus einem Programm heraus nicht bekannt sind.

Bei GFA-'.LST'-Files, also reinen ASCII-Dateien, ist das Format und die Art und Weise des Zugriffs durch den Interpreter bekannt. Aus diesem Grund können wir es uns leisten, diese Dateien im Rahmen der Möglichkeiten zu verändern.

Mit diesem 'LST_COMP.BAS' ist das ganz einfach.

Beim Einlesen von ASCII-Dateien durch die 'Merge'-Funktion überprüft der Interpreter die Richtigkeit der gelesenen Zeilen. Wenn man nun weiß, daß die Programmstruktur vom Interpreter immer erst dann aufgebaut wird, wenn die entsprechenden Programmteile im Editor angezeigt werden und diese optische Struktur für den Interpreter selbst eigentlich unwichtig ist, ist der Gedanke, daß man auf diese Struktur auch in den LST.-Files verzichten kann, gar nicht so weit weg. In der ASCII-Datei wird die optische Struktur einfach durch entsprechend viele SPACE-Zeichen (Chr$(32)) gebildet.

Um nun das Disketten-File zu kürzen, brauchen wir als erstes also nur diese SPACE's herauszuoperieren.

Zweitens kann an jedem Programmzeilen-Ende auf das Line-Feed-Zeichen (Chr$(10)) verzichtet werden. Dem Interpreter genügt zum Erkennen des Zeilenendes das Carriage-Return-Zeichen (Chr$(13)). Das allein kann bei umfangreichen Programmen mit mehreren tausend Zeilen schon einige KByte ausmachen.

Die dritte Möglichkeit, Platz einzusparen, besteht darin, alle Befehlsnamen, die auch durch Abkürzungen eingegeben werden können, durch diese Abkürzungen zu ersetzen. Bei diesen Befehlsnamen handelt es sich grundsätzlich um Befehle, die nur direkt am Zeilenanfang auftreten können. Sonstige Funktionen, die auch innerhalb einer Zeile stehen können, verfügen über keine Abkürzungen.

Der Interpreter erkennt beim Einlesen des '.LST'-Files diese Abkürzungen. Ein GFA-BASIC-Programm wird im Speicher generell als Mnemonic-Code abgelegt und nur bei der Bildschirm-Anzeige des Listings im Editor bzw. beim 'TRON'- und 'LIST'-Befehl in das für den Programmierer erkennbare Listing umgewandelt. Dieses Programm vollzieht nun diese drei Schritte und schreibt die dadurch komprimierte Datei wieder auf die Diskette. Im allgemeinen sind damit Platzeinsparungen von bis zu 30 Prozent möglich. Ein '.LST'-File von 100 KByte Länge hätte demnach hinterher eine Länge von nur 70-75 KByte.

```
' Programm: LST_CMPR.BAS
'
' |""""""""""""""""""""""""""""""""""""""""""""|
' |      '.LST' - FILE - KOMPRESSOR              |
' |----------------------------------------------|
' '''''''''''''''''''''''''''''''''''''''''''''''

On Break Gosub Schluss                  ! Abbruch abfangen
Lpoke Xbios(2)+32000,Fre(0)             !  aktuellen freien Speicherplatz
'                                       !   in den üblicherweise ungenutzten
'                                       !   freien Bereich hinter dem Bild-
'                                       !   schirmspeicher schreiben.
Reserve 50000                           !   50 KByte Reserve für BASIC
Deffill ,2,4
Pbox 10,10,629,389
Alert 2,">.LST-FILE-COMPRESSOR<",3,"DO IT",Flag%
Al$="Wurde das LST-File mit|Deflist 0 (Befehlsname) oder|"
Al$=Al$+"Deflist 1 (Befehlsname)|abgespeichert ?"
Alert 2,Al$,1,"Deflist1|DEFLIST0",D.efflag%
Restore Befehle                         ! Data-Zeiger setzen
Do                                      ! erste Lese-Schleife
  Read Dumm$                            !  Data lesen
  Exit If Instr(Dumm$,"XXX")            !  Exit, wenn Endmarkierung erreicht
  Inc D.atas%                           !  mitzählen
Loop
Div D.atas%,2                           ! Data-Anzahl / zwei, da immer zwei
'                                       !        Datas pro Befehl
Restore Befehle                         ! nochmal Data-Zeiger setzen
Dim Lang$(D.atas%),Kurz$(D.atas%)       ! Befehls-Array dimensionieren
For I%=1 To D.atas%                     ! Alle Befehle und Abkürzungen
  Read Lang$(I%),Kurz$(I%)              !   lesen
  If Defflag%=2                         !   Mit 'DEFLIST 0' abgespeichert?
    Lang$(I%)=Upper$(Lang$(I%))         !   Befehl in Großbuchstaben
  Endif
Next I%                                 ! nächster Befehl
Fileselect "\*.lst","",S$               ! zu komprimierende Datei wählen
If S$<>"" And S$<>".LST"                ! gültiger Dateiname?
  Open "I",#1,S$                        !  Datei öffnen
  Size%=Lof(#1)                         !  Dateilänge feststellen
  Bload S$,Himem                        !  Datei hinter BASIC-Speicher laden
  Print At(10,10);"Comprimiere Zeile : "
  Do                                    !  Kompressor-Schleife
    Count%=0                            !   Bytezähler auf Null
    A$=""                               !   Zeilenpuffer löschen
```

```
  Do                                  !  Leseschleife
    Byte%=Peek(Himem+Offset%+Count%)!    Einzelzeichen selektieren
    Inc Count%                        !    Bytezähler +1
    Exit If Byte%=13                  !    Exit, wenn CarriageReturn
    A$=A$+Chr$(Byte%)                 !    Zeile zusammenfügen
  Loop
  Add Offset%,Len(A$)+2               !  Positionszähler addieren
  '                                   !  (+2 bedeutet, daß die Zeichen
  '                                   !  CR und LF mitgezählt werden
  '                                   !  müssen)
  Exit If Offset%>Size%               !  Exit, wenn Ende des alten
  '                                   !  Listings
  Inc D.one%                          !  Zeilen mitzählen
  Print At(43,10);D.one%              !  aktuelle Zeile ausgeben
  Co%=0                               !  SPACE-Zähler auf Null
  Do                                  !  SPACE-Zählschleife
    Inc Co%                           !    SPACE-Zähler +1
    M$=Mid$(A$,Co%,1)                 !    Einzelzeichen selektieren
    Exit If M$<>" "                   !    Exit, wenn ungleich SPACE
  Loop                                !  Schleifen-Wende
  A$=Right$(A$,Len(A$)-(Co%-1))+Chr$(13) ! Restzeile rechts vom
  '                                   !   letzten SPACE bilden
  For I%=1 To D.atas%                 !  Befehlsliste durchgehen
    If Left$(A$,Len(Lang$(I%)))=Lang$(I%) ! 1. Wort in Liste gefunden?
      A$=Right$(A$,Len(A$)-Len(Lang$(I%)))!   Restzeile rechts vom
      '                               !       Befehlsnamen bilden
      A$=Kurz$(I%)+A$                 !       Abkürzung einsetzen
    Endif
  Next I%                             !  nächster Befehl der Liste
  Bmove Varptr(A$),Himem+Size%+Gesamt%,Len(A$) ! Zeile komprimiert
  '                                   !  zurück
  Add Gesamt%,Len(A$)                 !  bisher komprimierte Länge
  Exit If E.flag%=1                   !  Ende, wenn Exit-Flag gesetzt
Loop                                  ! große Schleifen-Wende
If E.flag%=1                          ! Exit-Flag gesetzt?
  Goto Raus                           !  Restprogramm überspringen
Endif
Startkill:                            !
Alert 2,"Altes LST-File löschen ?",1,"OKAY|NEIN",Back%
If Back%=1                            ! altes '.LST'-File löschen?
  If Exist(S$)                        !  Diskette nicht gewechselt?
    K.flag%=1
    Kill S$                           !    Datei löschen
  Else                                !  Diskette gewechselt!
    Al$="File nicht gefunden !|Diskette wurde gewechselt !"
    Alert 1,Al$,1,"Nochmal|Weiter",Back%
```

```
      If Back%=1                    !    Nochmal versuchen?
        Goto Startkill              !    dann zurück
      Endif
    Endif
  Endif
  Startsave:
  If Dfree(0)>Gesamt%+1000          ! Reicht Diskettenplatz aus?
    F.name$=Right$(S$,12)           !  Dateinamen selektieren
    Backslash%=Instr(F.name$,"\")   !   Backslash enthalten?
    F.name$=Right$(F.name$,Len(F.name$)-Backslash%)!Reststring=Dateiname
    If K.flag%=0                    !   altes File nicht gelöscht?
      F.name$=Left$(F.name$,Len(F.name$)-4)+".BAC" ! Extension '.BAC'
    '                                              ! setzen
    Endif
    Fileselect "\*.*",F.name$,F.name$   !
    If F.name$<>"" And F.name$<>"\"     !   gültiger Dateiname?
      Bsave F.name$,Himem+Size%,Gesamt% !     aus Speicher zurück auf
    '                                   !     Disk
      Inc C.opy%                        !     Kopien mitzählen
      Alert 2,Str$(C.opy%)+". KOPIE ablegen ??",1,"Nein|Okay",Back%
      If Back%=2                        !     Noch eine Kopie ?
        Goto Startsave                  !       dann nochmal speichern
      Endif
    Endif
  Else                                  ! Diskettenplatz reicht nicht
    Al$="Disketten-Speicherplatz reicht|nicht aus ! Evt. Disk-Wechsel !"
    Alert 1,Al$,1,"Nochmal|Abbruch",Back%
    If Back%=1                          !    Nochmal versuchen?
      Goto Startsave                    !      dann nochmal
    Endif
  Endif
Endif
Raus:
Clear                                   !    BASIC-Speicher klar
Reserve Lpeek(Xbios(2)+32000)           !    BASIC-Speicher wieder auf
'                                       !                 volle Größe
Edit                                    !    Programmende!
Procedure Schluss
  '
  ' Diese Break-Abfang-Routine ist notwendig, um bei Programmende den
  ' BASIC-Speicher wieder ordnungsgemäß restaurieren zu können.
  '
  E.flag%=1                             ! Abbruchflag setzen
  Cont                                  ! weiter geht's
Return
```

Die Data-Liste ist evtl. noch nicht vollständig. Sollten Ihnen noch weitere Abkürzungen einfallen, brauchen Sie diese einfach nur in diese Liste einzutragen. Dabei ist jedoch zu beachten, daß der ausgeschriebene und der abgekürzte Befehlsname immer als Paar (erst ausgeschrieben, dann abgekürzt) direkt hintereinanderstehen und daß als letztes Data-Element der String 'XXXXX' verwendet wird.

```
Befehle:
Data Pbox,pb,Return,ret,Repeat,rep
Data Procedure,pro,Print,p,Box,b,Until,u
Data Loop,l,For ,f ,Next,n,Fill,fi,Mouse,m
Data Put,pu,Text,t,Defline,de,Swap,sw
Data Line,li,Plot,pl,Local,loc,Draw,dr
Data Data,d,Defmark,defm,Defmouse,defmo
Data Deffill,deff,Deftext,deft,Dim,di
Data Erase,er,Endif,endi,Alert,a,Input,inp
Data Restore,res,Circle,c,Color,co
Data Setcolor,se,Pcircle,pc,Ellipse,ell
Data Pellipse,pe,Prbox,prb,Rbox,rb,Else,el
Data Poke,po,Lpoke,lp,Pause,Pa,Goto,got
Data Resume,resu,Read,rea,If,i,Inc,in
Data Add,ad,Sub,s,Rem,',Gosub,@
Data Reserve,rese,Bmove,bm,Showm,sh
Data Hidem,hi,Option Base,opt base
Data Option ",opt ",Void,vo,Vdisys,V
Data Void,vo,Gemsys,ge,Open ,o ,Edit,ed
Data Fileselect,filese,Graphmode,g,Bload,bl
Data Bsave,bs,Dpoke,dp,Monitor,mon,XXXXX
```

5.12 Alles Schiebung

Diese Prozedur stellt wieder ein, meines Erachtens nach, sehr wichtiges Utility zur Verfügung.

Es ist immer wieder notwendig, den Benutzer zu Werteeingaben aufzufordern. Diese Routine erledigt das. Es kann dabei ein bestimmter Wertebereich vorgegeben werden. Eingaben außerhalb dieses Bereichs sind also nicht möglich.

Des weiteren können zwei Textstrings übergeben werden, von denen der erste eine Überschrift bereitstellt (evtl. Eingabekommentar). Der zweite enthält die Bezeichnung für die einzustellende Werteeinheit (z.B. DM, Meter, kg etc.).

Als letztes muß eine Integer-Pointer-Variable übergeben werden, in welcher nach Prozedurende der tatsächlich eingestellte Wert zurückgeliefert wird.

Das Besondere an dieser Routine ist, daß sie eine Möglichkeit verdeutlicht, wie man grafische Objekte an die verschiedenen Bildschirm-Auflösungen anpassen kann. Diese Slide-Box ist absolut mit GEM-Formularen vergleichbar, da sie sich nach der aktuelle Auflösung richtet.

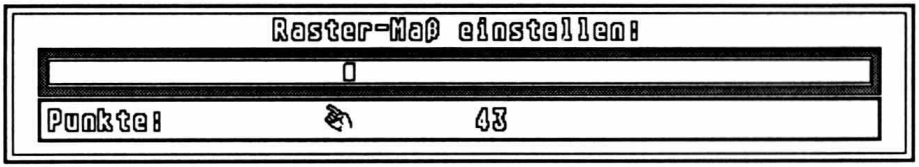


Abb. 36: GFA-Slider

```
' Programm: SLIDER.BAS
'
' |''''''
```

```
  Box 55/Xr%,145/Yr%,585/Xr%,229/Yr%                    !
  Text 320/Xr%-Int(Len(Sl_txt$)/2)*Tc%,160/Yr%,Sl_txt$ !
  Text 76/Xr%,214/Yr%,Einheit$                          !
  Deffill 1,2,4                                         !
  Pbox 70/Xr%,166/Yr%,570/Xr%,194/Yr%                   !-- Box-Aufbau
  Deffill ,0,0                                          !
  Pbox 75/Xr%,171/Yr%,565/Xr%,189/Yr%                   !
  Box 76/Xr%,173/Yr%,564/Xr%,187/Yr%                    !
  Box 70/Xr%,196/Yr%,570/Xr%,221/Yr%                    !
  Box 71/Xr%,197/Yr%,569/Xr%,220/Yr%                    !
  Graphmode 3                                     !----'
  Repeat                                          !----.
    Box 79/Xr%,175/Yr%,85/Xr%,185/Yr%                   !
    Repeat                                              !   Auf
    Until Mousek                                        !-- Mausklick
    Box 79/Xr%,175/Yr%,85/Xr%,185/Yr%                   !   warten
  Until Mousek                                    !----'
  Repeat                          !Start der Einstellungsschleife
    Mouse X%,Y%,K%                !  Mausstatus
    Y%=185/Yr%                    !   Y-Koordinate feststehend
    If X%>561/Xr%                 !   Maus rechts außerhalb?
      X%=561/Xr%                  !    X-Maximum setzen
    Endif
    If X%<85/Xr%                  !   Maus links außerhalb?
      X%=85/Xr%                   !    X-Minimum setzen
    Endif
    Pbox X%-6/Xr%,Y%-10/Yr%,X%,Y% !   Schiebe-Box zeichnen
    Sl%=Int(X%-(86-Xr%)/Xr%)/((476/Xr%)/(Ew%-Sw%))+Sw%
    '                             !   Wert berechnen
    Graphmode 1
    Text 320/Xr%-Int(Len(Str$(Sl%))/2)*Tc%,214/Yr%," "+Str$(Sl%)+"  "
    '                             !   Wert anzeigen
    Graphmode 3
    Repeat                        !   Warteschleife
      Mouse X1%,Y1%,K%            !    Mausstatus
    Until X1%<>X% Or K%=0         !   Weiter, wenn Mausbewegung
    '                             !   oder Maustaste frei
    Pbox X%-6/Xr%,Y%-10/Yr%,X%,Y% !   Schiebe-Box löschen
  Until K%=0                      !  Exit, wenn Maustaste frei
  *Rueckgabe%=Sl%                 !  Wert an Hauptprogramm zurück
  Put 50/Xr%,140/Yr%,Bg$          !  Hintergrund restaurieren
Return
```

5.13 Gut sortiert

Es werden immer wieder die raffiniertesten Sortier-Methoden vorgestellt. Sie haben meistens den großen Vorteil der Zeitersparnis. So ist eine Quicksort-Routine bis zu 10mal schneller als ein vergleichbares Bubblesort. Ich bezweifle jedoch, daß der größte Teil unserer Leser dermaßen große Datenmengen zu sortieren hat, daß der Geschwindigkeitsvorteil von erheblicher Bedeutung wäre. Der große Nachteil der Quicksort-Routinen ist nämlich die wesentlich erschwerte Durchschaubarkeit. Man muß sich schon intensiv mit Algorithmen dieser Art auseinandersetzen, um sie nach individuellen Bedürfnissen ändern zu können.

Deshalb möchte ich hier eine einfache Sortier-Routine vorstellen, die zwar nicht mit Turbo-Effekten glänzt, aber aufgrund ihrer Einfachheit leicht von Ihnen modifiziert werden kann.

Damit das Beispiel auch einen Nutzeffekt hat, wird zuerst das Inhaltsverzeichnis der aktuellen Diskette in eine Diskettendatei geschrieben, aus dieser wieder ausgelesen, sortiert und zum Schluß als Datazeilen sortiert wieder auf Diskette geschrieben.

Die Datei kann dann mit 'Merge' in den Arbeitsspeicher geladen und mit 'Llist' oder 'Block'-"L"list auf dem Drucker ausgegeben werden.

```
' Programm: SORTER.BAS
'
' |''''''
```

```
Close #1                                  ! File schließen
@Sorter(*A$(),I%-1)                       ! Feld sortieren
@Write(*A$(),I%-1,"filedat.lst")          ! sortiertes Feld auf Disk
'                                         !              schreiben
Procedure Sorter(Ptr%,Lim%)
```

Dies ist die erwähnte Sortier-Routine. Als Parameter werden ihr bei Aufruf der Pointer auf das zu sortierende Stringfeld und das Limit (letzter Feldindex) übergeben.

Beispiel: @Sorter(*A$(),I%-1)

Nach Änderung der Stringvariablen auf numerische Variablen, Löschen der beiden 'Upper$'-Zeilen und Änderung der If-Abfrage auf die zu vergleichenden Feld-Elemente

Beispiel: If Feld%(J%)=>Feld%(K%)

kann diese Routine auch zum Sortieren von numerischen Feldern verwendet werden.

```
  Local J%,K%,D1$,D2$
  Erase Puffer$()                 ! Feld löschen
  Dim Puffer$(Lim%)               !  Sortier-Buffer-Feld einrichten
  Swap *Ptr%,Puffer$()            !  Global- mit Local-Feld vertauschen
  For J%=1 To Lim%                !  alle Feld-Elemente durchgehen
    For K%=J%+1 To Lim%           !   nochmal alle Feld-Elemente, da ja
      '                           !   jedes Element mit jedem anderen
      '                           !   Element verglichen werden muß.
      '                           !   Der Start-Index dieser zweiten
      '                           !   Schleife wird bei jedem Durchlauf
      '                           !   um 1 erhöht, da die davorliegenden
      '                           !   Elmente schon sortiert sind.
      D1$=Upper$(Puffer$(J%))     !    Die beiden zu vergleichenden
      D2$=Upper$(Puffer$(K%))     !    Elemente in Großschrift umwandeln,
      '                           !    da sonst Kleinbuchstaben Priorität
      '                           !    vor Großbuchstaben erhalten.
      If D1$=>D2$                 !    Ist das erste Vergleichselement
        '                         !    größer oder gleich dem zweiten?
        Swap Puffer$(J%),Puffer$(K%)!   Element-Inhalte vertauschen
        '                         !      Ab jetzt wird mit einem anderen
        '                         !      Inhalt des Vergleichs-Elements
      Endif                       !                 weitergesucht.
```

```
    Next K%                          !  nächstes zweites Element
  Next J%                            ! nächstes Vergleichs-Element
  Swap *Ptr%,Puffer$()               ! nach Sortier-Ende wieder Local-
  '                                  ! mit Global-Feld vertauschen.
Return
Procedure Write(Ptr%,Lim%,Dat$)
```

Diese Prozedur erledigt das Ablegen der einzelnen Feld-Elemente als Data-Zeilen in eine Disketten-Datei. Diese Prozedur kann ebenfalls allgemein verwendet werden. Haben Sie also ein Feld, das Sie bewahren wollen, können Sie es so auf Disk schreiben und die hiermit produzierten Data-Zeilen hinterher weiterverarbeiten.

Als Parameter erwartet die Prozedur ebenso wie die Sortier-Prozedur den Feld-Pointer des betreffenden Feldes, den Index des letzten Elementes und zusätzlich den Namen der Datei, in welche die Daten geschrieben werden sollen.

```
  Local J%
  Erase Puffer$()                    !  Feld löschen
  Dim Puffer$(Lim%)                  !  Buffer-Feld einrichten
  Swap *Ptr%,Puffer$()               !  Global- mit Local-Feld vertauschen
  Open "O",#99,Dat$                  !  Datei öffnen
  For J%=1 To Lim%                   !  alle Feld-Elemente durchgehen
    Print #99,"D ";Puffer$(J%)       !    Inhalt in Datei schreiben
  Next J%                            !  nächstes Element
  Close #99                          !  Datei schließen
  Swap *Ptr%,Puffer$()               !  Local- m. Global-Feld zurücktauschen
Return
```

5.14 GFA-BASIC-Erweiterung: Sortier-Funktion per Monitor()

Zu guter Letzt wollen wir nicht versäumen, eine Anwendung für den bisher stiefmütterlich behandelten 'Monitor'-Befehl vorzustellen. Da dieser Befehl nur in Verbindung mit der Maschinensprache seine volle Blüte erlangt, kommen wir um einen Abstecher in Assembler nicht herum.

Stefan Dittrich, bekannter Autor von Maschinensprachentips für den ST, hat uns die folgenden Routinen samt ihrer Erläuterung zur Verfügung gestellt:

Ein BASIC-Interpreter wie GFA-BASIC 2.0 hat oft so viele Befehle und Funktionen, daß man recht lange braucht, bis man diese alle im Griff hat. Doch irgendwann kommt meist der Moment, in dem man die eine oder andere Funktion vermißt. So auch im GFA-BASIC, bei dessen Version 2.0 jedoch glücklicherweise eine Hintertür geöffnet wurde, um Erweiterungen zu entwickeln.

Die Rede ist hier von dem etwas ominösen MONITOR(n)-Befehl, für den im Handbuch auch kein Anwendungsbeispiel gegeben ist. Mit diesem Befehl kann man ein Maschinenprogramm aufrufen, dem auch einige Parameter übergeben werden.

Der am einfachsten zu verwendende Parameter ist derjenige, den man beim MONITOR(n)-Befehl direkt übergibt. Für n kann eine beliebige Integerzahl oder -Variable eingesetzt werden, deren Wert das Maschinenprogramm im Register D0 erhält.

Um das Maschinenprogramm aber überhaupt als Erweiterung zu initialisieren, bedarf es einiger Vorbereitungen:

Zuerst muß das Erweiterungsprogramm gestartet werden, und zwar vom Desktop aus. Dieses Programm muß sich nun den benötigten Speicherplatz reservieren und den Rest freigeben, damit das GFA-BASIC auch Speicher zur Verfügung hat.

Dann muß der Illegal-Instruction-Vektor (ab $10) auf die Routine im Programm umgebogen werden, welche die eigentliche Befehlserweiterung darstellt.

Nun kann entweder z.B. mit KEEP PROCESS das Programm verlassen werden, wodurch es im Speicher bleibt, und GFA-BASIC gestartet werden. Oder aber das Programm lädt und startet direkt den GFA-Interpreter, was die bequemste Art ist.

Diese Methode verwendet auch das hier vorgestellte Programm. Dazu muß der Interpreter natürlich zusammen mit dem Programm auf einer Diskette sein.

Die Befehlserweiterung selbst ist ein einfaches Sortierprogramm, welches ein eindimensionales String-Array nach nur einem Buchstaben sortiert. Es ist mehr als Demonstration sowohl des MONITOR-Befehls als auch der Variablen-Speicherung bzw. -Struktur des GFA-Interpreters gedacht, leistet jedoch für so manchen Zweck gute und recht schnelle Dienste. Außerdem hat es mich geärgert, daß in der GFA-Beschreibung so lapidar die Verwendungsmöglichkeit des MONITOR-Befehls als Aufruf einer Sortier-Funktion vorgeschlagen wird, ohne einen Hinweis, wie dies zu bewerkstelligen ist. Deshalb nun das vorliegende Programm.

Das Programm ist in zwei Teile unterteilt: die Initialisierung und die eigentliche Erweiterung, die Sortier-Funktion.

Die Initialisierung läuft in 3 Schritten ab:

1. Einrichtung eines neuen Stacks und Reservierung des benötigten Speicherplatzes

2. Aufruf einer Routine im Supervisor-Modus, die den Illegal-Instruction-Vektor auf die Sortier-Routine ab dem Label 'main' umbiegt

3. Laden und Starten des GFA-BASIC-Interpreters mittels der EXECUTE-Funktion des GEMDOS.

Außerdem enthält es noch die TERM-Funktion des GEMDOS, die nach Rückkehr aus dem Interpreter (Quit) das System wieder ans Desktop übergibt.

Hier nun der erste Teil des Programmes:

```
;Programm: GFA_SORT.S

;** Befehlserweiterung für GFA-BASIC   V1.0  S.D. **
;** Sortierung eines 1-dimensionalen Stringfeldes **
;**              Aufruf: MONITOR(*a$())           **

run:
 move.l 4(sp),a5
 lea stack,sp        ;neuen Stack einrichten

 move.l #ende-run+$500,-(sp) ;Speicher reservieren
 move.l a5,-(sp)     ;Speicher-Anfang
 clr -(sp)
 move #$4a,-(sp)     ;Setblock-Funktion
 trap #1
 add.l #12,sp

 pea init            ;Vektorverbiegung
 move #$26,-(sp)     ;mit Super_Execute
 trap #14            ;ausführen
 addq.l #6,sp

 pea param           ;Environment
 pea param           ;Kommandozeile
 pea fname           ;Filename: GFABASIC.PRG
 clr -(sp)           ;Load and Go
 move #$4b,-(sp)     ;EXECUTE:
 trap #1             ;GFA-BASIC starten

 add.l #16,sp        ;Rückkehr erst nach GFA-'Quit'

exit:
 clr -(sp)
 trap #1        ;Term => Desktop

init:           ;Zeiger verbiegen
 move.l #main,$10 ;Illegal-Pointer auf eigene Routine
 rts            ;fertig

param: dc.w 0  ;kein Environment oder Kommandozeile
fname: dc.b "GFABASIC.PRG",0
even
```

Nun zum etwas komplizierteren Teil: der Sortier-Routine. Hier wird vorsichtshalber erst einmal getestet, ob auch der Zeiger auf

ein eindimensionales Feld übergeben wurde. Ist dies nicht der Fall, so wird in die 'error'-Routine verzweigt, welche nach Ausgabe des Glockentones (CHR$(7)) wieder zum GFA-Programm zurückgibt.

Ist es doch ein richtiges Feld, so wird die Adresse des Feldes selbst in Adreßregister A0 geladen. Schließlich zeigt *A$() nur auf den Feld-Descriptor, nicht auf das Feld selbst.

Danach wird die Anzahl der Felder in D0 geladen. Für die folgende Schleife wird dieser Wert zuerst um 2 verringert. Danach wird das Tausch-Flag (D3) gelöscht. Dieses gibt nach Durchlauf der inneren Schleife (lop1) an, ob fertig sortiert ist oder nicht.

Das Register D1 wird nun mit 4 geladen. Der darauffolgende Befehl move.l 0(a0,d1),a1 lädt nun die Adresse des ersten Feldes (z. B. A$(0)) in A1. Ebenso wird die Adresse des nächsten Feldes (A$(1)) in A2 gelegt.

Die an diesen beiden Speicherstellen liegenden Bytes, eben die Anfangsbuchstaben der Strings A$(0) und A$(1) werden nun verglichen. Ist A$(1) größer oder gleich A$(0), so wird die Schleife bei 'noswap' fortgeführt. Ist A$(1) jedoch kleiner, so müssen a$(0) und A$(1) vertauscht werden.

Dies erwies sich als der schwierigste Teil des ganzen Programmes. Da die beiden zu vertauschenden Strings ja nicht unbedingt gleich lang sein müssen, kann die einfache Vertauschung der Variableninhalte nicht in Frage. Also werden statt dessen die zugehörigen Zeiger und Längenangaben vertauscht.

Zuerst die Adressen: Hierfür müssen nur die Adreßregister A1 und A2 wieder zurückgeschrieben werden, natürlich über Kreuz.

Danach die Längen: Auch diese Werte werden zuerst in zwei Register (D4 und D5) ausgelesen und nachher zurückgeschrieben. Diese etwas umständliche Art ist deshalb sinnvoll, da wir die Längen noch brauchen. Denn jetzt kommen die Backtrailer: Sie liegen hinter jedem String, und zwar immer an einer geraden Adresse. Also müssen die Längen, die ja als Offset vom String-

anfang verwendet werden müssen, ggf. auf einen geraden Wert erhöht werden. Dies geschieht durch die beiden Befehle addq #1,d4 und and #$fffe,d4.

Mit dem so erhaltenen Offset kann nun mittels move.l 0(a1,d4),d6 der Backtrailer des ersten Strings (A$(0)) ermittelt und in D6 geladen werden. Nach Übertragung des zweiten in den ersten Backtrailer-Wert wird D6 in den zweiten geschrieben.

Fertig ist die Vertauschung. Nun wird noch der Offset D1 erhöht, damit auf den nächsten String zugegriffen werden kann, und das Tausch-Flag D3 gesetzt, welches nach Ablauf der inneren Schleife mit tst d3 abgefragt wird. Ist es dann nicht null, so wird die äußere Schleife wiederholt (loop), andernfalls ist der Sortiervorgang abgeschlossen. Dann erst wird wieder zum GFA-Interpreter übergeben.

Hier nun die beschriebene Routine zum aufsteigenden Sortieren eines eindimensionalen String-Arrays nach einem signifikanten Buchstaben:

```
;** Hauptroutine: Sortierung eines String-Arrays  S.D. **

main:
     move.l    d0,a0      ;Parameter in A0
     cmp       #1,4(a0)   ;eindimensional ?
     bne       error      ;nein !
     move.l    (a0),a0    ;Adresse des Feldes in A0
     clr.l     d4         ;vorsichtshalber löschen
     clr.l     d5

;* äußere Schleife *

loop:
     move.l    (a0),d0    ;Anzahl der Felder ermitteln
     subq      #2,d0      ;und Zahl korrigieren

     clr       d3         ;Tausch-Flag löschen
     moveq     #4,d1      ;D1 vorbereiten

;* innere Schleife *
```

```
lop1:
     move.l    0(a0,d1),a1 ;Adresse von A$(0) ect.
     move.l    6(a0,d1),a2 ;Adresse von A$(1) ect.
     move.b    (a2),d2
     cmp.b     (a1),d2          ;Vergleich
     bpl       noswap ;OK

     move.l    a1,6(a0,d1)
     move.l    a2,0(a0,d1)      ;Adressentausch

     move      4(a0,d1),d4      ;Länge 0
     move      10(a0,d1),d5     ;Länge 1
     move      d5,4(a0,d1)
     move      d4,10(a0,d1)     ;Längentausch

     addq      #1,d4
     and       #$fffe,d4        ;Längen ggf. aufrunden
     addq      #1,d5
     and       #$fffe,d5

     move.l    0(a1,d4),d6      ;Backtrailer 0, Backtrailertausch
     move.l    0(a2,d5),0(a1,d4) ;BT1 in BT0
     move.l    d6,0(a2,d5)      ;BT0 in BT1

     st        d3               ;Flag setzen

noswap:
     addq.l    #6,d1            ;Offset erhöhen
     dbra      d0,lop1          ;weiter
     tst       d3               ;fertig ?
     bne       loop             ;nein: nochmal

stop:
     addq.l    #2,2(sp)         ;PC auf Befehl nach ILLEGAL richten
     rte                        ;zurück zu GFA-BASIC

error:
     move      #7,-(sp)
     move      #2,-(sp)
     trap      #1          ;Glocke ertönen lassen: falscher Parameter!
     addq.l    #4,sp

     bra stop              ;Rückkehr zum GFA-Interpreter
```

```
ende:

data
        blk.l $100
stack: blk.l 1
```

Hier nun noch ein kleines Beispiel-Programm in GFA-BASIC, mit dem die Sortier-Routine demonstriert werden kann. Das anfängliche Auffüllen der Strings dient zur Sicherheit, damit nach dem Sortieren nicht falsche Daten auftauchen.

```
' Programm: GFA_SORT.BAS
'
' *** Demo-Programm für die Sortier-Routine  S.D. ***
'
Dim A$(10)              !Feld dimensionieren
For I%=0 To 10
  A$(I%)="!"             !vorsichtshalber auffüllen
Next I%
'
A$(0)="Stefan"          !Strings belegen
A$(1)="Dittrich"
A$(2)="GFA-BASIC"
A$(3)="Erweiterung:"
A$(4)="Sortieren"
A$(5)="von"
A$(6)="indizierten"
A$(7)="Strings"
A$(8)="für 'TIPS"
A$(9)="UND TRICKS'"
A$(10)="zu GFA-BASIC"
'
Monitor (*A$())         !sortieren...
'
For I%=0 To 10
  Print A$(I%)           !und ausgeben
Next I%
```

Anhang A: Verzeichnis der TOS- und GEM-Routinen

XBIOS-Funktionen (eXtendedInputOutputSystem)

XBIOS(2) (physikalische Bildschirmadresse) 116
XBIOS(3) (logische Bildschirmadresse) 116
XBIOS(4) (aktuelle Bildschirmauflösung) 116
XBIOS(5) (Logbase und Physbase ändern) 117
XBIOS(6) (Farbpalette neu setzen) 118
XBIOS(7) (Fargregister lesen) 119
XBIOS(8) (Sektoren lesen) 119
XBIOS(9) (Sektoren schreiben) 119
XBIOS(10) (Tracks formatieren) 120
XBIOS(12) (MIDI-String ausgeben) 122
XBIOS(14) (I/O-Pufferadressen) 123
XBIOS(15) (RS232-Konfiguration) 123
XBIOS(16) (Tastaturbelegung setzen/ermitteln) 124
XBIOS(17) (Zufallswert) 126
XBIOS(18) (Bootsektor erzeugen) 127
XBIOS(19) (Sektoren verifizieren) 120, 132
XBIOS(21) (TOS-Cursor-Attribute) 132
XBIOS(24) (Tastaturbelegung restaurieren) 133
XBIOS(26) (MFP-Interrupt sperren) 133
XBIOS(27) (MFP-Interrupt zulassen) 133
XBIOS(28) (Soundregister lesen/setzen) 134
XBIOS(29) (Soundchip-Floppy-Port-A-Bits setzen) 135
XBIOS(30) (Soundchip-Floppy-Port-A-Bits löschen) 135
XBIOS(32) (Soundstring im Interrupt) 136
XBIOS(33) (Druckerkonfiguration setzen/lesen) 138
XBIOS(35) (Tastatur-Repeat setzen/lesen) 138
XBIOS(38) ('Call' im Supervisor-Modus) 139
XBIOS(39) (Disk-TOS-AES ausschalten) 139

GEMDOS-Funktionen
(GraphicEnvironmentManager DiskOperatingSystem)

GEMDOS(1)	(offene Einzelzeichen-Eingabe)	140
GEMDOS(7)	(verdeckte Einzelzeichen-Eingabe)	140
GEMDOS(25)	(aktuelles Laufwerk ermitteln)	141
GEMDOS(26)	(DiskTransferAdresse setzen)	142
GEMDOS(32)	(Supervisor-Modus an/aus)	143
GEMDOS(47)	(DiskTransferAdresse ermitteln)	144
GEMDOS(67)	(File-Attribute ändern)	145
GEMDOS(72)	(freien Speicher lesen/reservieren)	146
GEMDOS(73)	(reservierten Speicher freigeben)	148
GEMDOS(74)	(Speicherblock reservieren)	148
GEMDOS(78)	(Datei suchen)	149
GEMDOS(79)	(weitere Dateien suchen)	150

BIOS-Funktionen (BasicInputOutputSystem)

BIOS(4)	(Sektoren lesen/schreiben)	151
BIOS(7)	(Bios-Parmeter-Block-Adresse)	152
BIOS(9)	(Diskettenwechsel feststellen)	153
BIOS(10)	(angeschlossene Laufwerke ermitteln)	153
BIOS(11)	(Umschalttasten-Status lesen/setzen)	153

VDISYS-Routinen

GETHANDLE	(VDI-Handle ermitteln)	156
VDISYS 164	(Desktop-Linienart u. -dicke ändern)	164
VDISYS 16	(Desktop-Linienart u. -dicke ändern)	164
VDISYS 113	(Desktop-Linienart u. -dicke ändern)	164
VDISYS 12	(Desktop-Textart, -höhe, -winkel ändern)	164
VDISYS 13	(Desktop-Textart, -höhe, -winkel ändern)	164
VDISYS 106	(Desktop-Textart, -höhe, -winkel ändern)	164
VDISYS 35	(aktuelle Linien-Attribute ermitteln)	160
VDISYS 36	(aktuelle Marker-Attribute ermitteln)	160
VDISYS 37	(aktuelle Füll-Attribute ermitteln)	160
VDISYS 38	(aktuelle Text-Attribute ermitteln)	161
VDISYS 104	('P'-Umrahmung an/aus)	156
VDISYS 114	(randlose Pbox zeichnen)	157
VDISYS 129	(Grafik-Ausgabebereich bestimmen)	159

GEMSYS-Routinen

GEMSYS 52 (Alertbox produzieren) 169
GEMSYS 70 ('Gummiband'-Box produzieren) 165
GEMSYS 71 ('Schiebe'-Box produzieren) 166
GEMSYS 72 (Box-Bewegungseffekt produzieren) 167
GEMSYS 73 (Box-Öffnungseffekt produzieren) 168
GEMSYS 74 (Box-Schließungseffekt produzieren) 168

Anhang B: Stichwortverzeichnis

Adreßregister 338
AES-Icon-Speicher 38
AES-Icons 36
Aktionspunkt 27, 32, 43, 53
Alert-Icons 40, 49, 67
Anfangs-CLS 73
APPLBLK 181
Auflösung 116

Backup 321
BAS-HEADER 301
Baudrate 124
Bildschirm-Speicher 16f, 60, 116
Binärdatas 50, 62
Bios-Parameter-Block 92, 152
BITBLK 202, 269
Bitmuster 22, 39
Boot-Sektor 87, 91, 100, 108, 121, 130

Clipping 157, 245
Cluster 90
Control-Feld 158
Cursor 132
Cursor-Blinkfrequenz 132

Deffn 115
Desktop-Icons 33, 41, 67
Diodenkabel 122
Directory 89, 128
Directory-Cluster 98
Disk-Puffer 82
Disk-Transfer-Adresse 142
Disketten-TOS 24, 65
Dreieckwelle 137
Druckeranpassung 281
Druckereinstellung 138

EDITOR 42
EXIT-Objekt 199
Extension 28, 321

Farbpalette 80
Farbregister 60, 118
FAT 88, 92, 102, 121, 129
FORM ALERT 169
Formular 198
Form_do 199
Funktionstastenbelegung 222

GEM-Objektbäume 171
Get/Put-String 59, 61
GRAF GROWBOX 168
GRAF MOVEBOX 167
GRAF SHRINKBOX 168
Graf_dragbox 227
Graf_rubberbox 227

Hardcopy 82
Header 189
Hidden_line_Algorithmus 289

ICONBLK 181, 209, 268
Icons 209
Illegal-Instruction-Vektor 335
Image 207
Interrupt-Routinen 79

Joystick-Aktionen 70

Laufwerksnummer 78
Lesekopf-Bewegung 77

Makros 113
Maus-Icon 32, 38, 40
Maus-Icons austauschen 52
Mausbewegung 69, 72
Maushintergrund 73, 96

Mausindex 55
Mausmaske 53
Media-Change 95
Media-Change-Flag 68
Menü 229
Menu_bar 230
Menu_ienable 230
Menu_text 231
Menu_tnormal 231
MIDI-Ports 122
MONITOR(n) 335
Muster-Speicherung 46

Nibbels 104

Objc_change 195
Objc_draw 185
Objc_find 224
Objektbaum 171
Objekte 171
Objekteigenschaften 176
Objektindex 198
Objektspezifikation 179f, 264
Objektstruktur 171
Objektzustand 177

Parity 124
Präsentationsgrafiken 289
Prozessor-Register 81

Quader 290

RADIO BUTTON 187
RAM-Adressen 68
RCS 205, 213, 258
Rechteckliste 244
REDRAW 244, 252
Rekursionen 317
Rekursiv 40
Reset-Vektor 77

Resource Construction Set 186, 229
Resources-File 172
RS232 ..... 123, 282
RSC 191, 192
Rsrc_free 192
Rsrc_gaddr 193
Rsrc_load 192

Sägezahn 137
Schieber 238
Schnappschuß 25
Scroll-Routinen 249
Seitenflächen 290
Sektoren 120, 129, 151
Skalierung 71
Slider 241, 249
Snapfile 31
Sonderstruktur 172
Sondertasten-Status 71
Sound-Register 134, 138
Suchstring 35
Supervisor-Modus 74, 139, 143, 336

Tastatur-Repeat 81, 139
Tastaturbelegung 124, 133
Tastaturpuffer 25
TEDINFO 174, 181, 196, 197, 265, 268
Tortendiagramm 298
Track 92, 121, 129, 151
TREE 232

Uhrzeit 287
Unterdirectory 109
User-Modus 143

VBL-Routinen 79ff
Verify 78

Wechseltasten 153
Wildcards 150
Window_handling 236
Windtab 233
Windtab-Tabelle 255
Wind_create 255
Wind_get 247
Wind_open 256
Wind_set 224, 256
Wi_handle% 237
Wurzelobjekt 174, 259

Xor 18, 25

Zeichenbox 159
Zeichenbreite 159
Zeichensatz-Speicher 65

Verlassen Sie sich nicht auf Gerüchte, profitieren Sie lieber vom DATA BECKER-Know-How. Keine nackten Befehlsübersichten, sondern wirklich brauchbares Material liefert dieses Buch in Hülle und Fülle. Dem Autor ist es erstmalig gelungen, ein wirklich spannendes Buch zum neuen GfA-BASIC zu schreiben. So lernen Sie nicht nur schnell und kompetent das GfA-BASIC und die Systemprogrammierung des ATARI ST kennen, sondern erhalten zusätzlich noch ein komplettes GRAPHIC CONSTRUCTION SET, das mit den gebotenen Features garantiert einmalig ist.

Aus dem Inhalt:

- Kommandos des GfA-EDITORS
- Strukturierte Programmierung kontra Spaghetti-Code
- Professionelle Programmerstellung anhand des GCS
- Trickfilm-Modus in Echtzeit
- Patch-Grafik-Design
- 3-D-Animationsgrafik
- Pattern- und Grafik-Editor
- Verzerren, Spiegeln, 3-D-Trommel-Generator
- Clipping, Blockverarbeitung, Netzmodus
- 2 aktive/2 passive Grafikseiten und vieles mehr

Litzkendorf
Das große GfA-BASIC-Buch
Hardcover, 468 Seiten, DM 49,–
ISBN 3-89011-222-6

Die Grafikfähigkeiten des ST gezielt für eigene Anwendungen einsetzen – dieses Buch zeigt Ihnen, wie es geht. Mit den vielen Beispielprogrammen in GFA-BASIC, C und Assembler, deren Source-Code komplett auf der Diskette zum Buch mitgeliefert wird, bestimmen Sie den Schwierigkeitsgrad selbst. So ist Ihr Erfolg vorprogrammiert.

Aus dem Inhalt:

- Bildschirmfenster und Grafikausgaben
- Grundlegende Grafikroutinen
- Mausverwaltung
- Sprite-Programmierung
- Poster-Hardcopy
- 3-D-Grafik und CAD
- Bewegte Bilder in 3-D
- Laufschriften
- Spiele
- Funktionsdarstellungen
- Busineß-Grafik
- Statistik
- GDOS und Metafiles
- Trickfilmproduktion mit Super-8 und Video
- Grafikmanipulationen
- Programmierung des Rasterzeileninterrupt
- Flackerfreie Animation
- Anschluß fremder Monitore
- Ein-/Ausgabe von Grafiken auf Diskette
- Druckeransteuerung
- Viele Grafikalgorithmen

Plenge
Das Supergrafik-Buch zum Atari ST
383 Seiten, DM 69,–
ISBN 3-89011-004-5

DAS STEHT DRIN:

GFA-BASIC ist ohne Zweifel eine der leistungsfähigsten BASIC-Versionen, die es für den ATARI ST gibt. Nur – wer diese fantastischen Fähigkeiten wirklich voll ausschöpfen will, braucht entsprechendes Know-how, braucht bei der Programmierung all die hilfreichen Tips & Tricks der GFA-Profis. Und damit der Spaß nicht beim Abtippen aufhört, werden alle Beispielprogramme gleich auf Diskette mitgeliefert.

Aus dem Inhalt:

- Komfortable Form-Input-Routine für eigene Maskenerstellung
- Hilfsroutinen für eigene Textverarbeitungen – WordWrap
- Taschenrechner als Hilfsprogramm mit Dezimal/Hexadezimal-Ausgabe
- Schnelle Sortier-Routine in Assembler zum Einbinden in eigene Dateiverwaltungen
- Die Routinen des Betriebssystems voll im Griff – GEMDOS, BIOS, XBIOS
- Neue Grafikroutinen durch Nutzung des VDI und AES
- Diskettenaufbau und Floppy-Programmierung, incl. Diskmonitor
- Super-Icon-Editor zum Erstellen von individuellen Desktop-Symbolen
- In die Tiefen des Rechners geschaut – die Speicherlupe
- Professionelle 3-D-Business-Grafik zur eindrucksvollen Präsentation
- GEM-Programmierung mit GFA-BASIC: die eigene Benutzeroberfläche, Pull-Down-Menüs, Windows, Eingabeboxen und beliebige Grafiken
- Das schier Unmögliche – 7 Fenster gleichzeitig öffnen, ohne daß es zieht
- Resource Construction Set in GFA-BASIC als Baukasten

UND GESCHRIEBEN HABEN DIESES BUCH:

Uwe Litzkendorf, Architekturstudent und Autor des großen GFA-BASIC-Buches, hat als einer der ersten mit dem GFA-BASIC gearbeitet. Sein Programmierhandbuch zum GFA-BASIC ist mittlerweile der Bestseller zu diesem Thema. Udo Onnen, Architekt und bekannter DATA WELT-Autor, ist passionierter BASIC-Programmierer und hat sich im Laufe der Zeit eine riesige GFA-System-Bibliothek aufgebaut, die er in diesem Buch erstmalig einem großen Anwenderkreis zur Verfügung stellt.

ISB N 3-89011-193-9 DM 49.00

DM 49,–
ÖS 382,–
sFr 47,–

DATA BECKER

9 783890 111933

Scan von Wosch